Vysoká škola báňská - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Implementácia relačnej databáze s dynamickým rozhraním pomocou ActionScriptu 3.0

Relational Database Implementation with Dynamic GUI in ActionScript 3.0

2012 Štefan Blanár

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Štefan Blanár**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Implementace relační databáze s dynamickým rozhraním pomocí AS3
Relational Database Implementation with Dynamic GUI in AS3

Zásady pro vypracování:

Cílem práce je popsat možnosti realizace informačního systému za pomocí Action Scriptu 3. Součástí práce je návrh a implementace takového systému, který bude vytvořen jako interaktivní aplikace vytvořená v AS3 využívající standardní možnosti při získávání, zpracování a ukládání dat z databáze.

Hlavní body práce:
1. Navrhněte IS a popiště jej k tomu vhodně zvolenými prostředky.
2. Nastudujte problematiku ActionScript 3 s důrazem na OOP přístup při vytváření aplikací a popište základní aspekty programovacího jazyka AS3.
3. Nastudujte a popište možnosti propojení báze dat s AS3.
4. Implementujte interaktivní rozhraní pro informační systém pomocí AS3, které bude propojeno s databází a váš postup a zvolené metody při vytváření IS popište.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Vích**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Chcel by som sa týmto poďakovať môjmu vedúcemu bakalárskej práce
Ing. Lukášovi Víchovi za jeho trpezlivosť, čas, úsilie, ochotu, pomoc a dôležité rady, ktorými mi pomohol pri vypracovaní tejto bakalárskej práce.

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 02.05. 2011 .....

Abstrakt :

Cieľom tejto práce je popísanie možností prepojenia báze dát a jazyka ActionScript 3.0, taktiež súčasťou práce je porovnanie jazyk ActionScript 3.0 s inými Objektovo Orientovanými programovacími jazykmi ako sú najmä C# a Java a následne nadobudnuté poznatky využiť v praxi pri implementácií demonštračného informačného systému využívajúceho štandardné možnosti pri získavaní, spracovaní a ukladaní dát do a z databáze za pomoci ActionScriptu 3.0.
Prvá časť tejto práce sa zaoberá samotným jazykom ActionScript 3.0 a jeho popisom.
V druhej časti sa podrobne venujem objektovo orientovaným programovacím princípom v jazyku ActionScript ako sú dedičnosť, polymorfiszmus, zapúzdrenie a využívanie menných priestorov. Osobitne sa tu venujem dvom rozličným spôsobom dedičnosti v ActionScripte 3.0 a rôznym typom objektov, ktoré sa tu vyskytujú.
V tretej časti sa zaoberám problematikou prepojenia ActionScriptu s relačnými databázami a z toho rezultujúcim problémom zabezpečenia komunikácie a presunu dát medzi ActionScript 3.0 a inými objektovo orientovanými programovacími jazykmi.
Napokon v poslednej časti sa venujem podrobnej dokumentácií demonštračného informačného systému na ktorom sú všetky spomínané princípu overené v praxi.

Kľúčové slova:

ActionScript, ASP.Net, C#, databáza(báza dát), dedičnosť, ECMAScript, Get metóda, Java, JSP, namespace, objeckt, OOP, PhP, polymorfizmus, Post metóda, query string, serverová(server-side) technológia, socket, XML

Abstract:

The main goal of this project is to describe possibilities of connectivity between database and ActionScript. Secondary objectives are description of AS 3.0 as Object oriented programming language and his comparison with other OOP languages such as Java or C# and then creation of demonstrative information system using all knowledge learned during these tasks.
First chapter is description of language ActionScript.
Second chapter describes ActionScript as object oriented programming language. This chapter takes specially attention on inheritance, polymorphism, encapsulation and using namespaces. Especially I describe here two different ways of inheritance in ActionScript and different types of objects used in AS.
The third part is about connection to database from AS 3.0 through other programming language such as Java, C# or PhP. In this part I solve integration between AS 3.0 and other OOP languages.
Last part of this document contains documentation for my demonstrational informational system, in which I test all of theory, which is described in this document.

Key words:

ActionScript, ASP.Net, C#, database, inheritance, ECMAScript, Get method, Java, JSP, namespace, object, OOP, PhP, polymorphism, Post method, query string, server-side technology, socket, XML

Slovník použitých pojmov, skratiek a symbolov :

AS – *ActionScript* objektovo orientovaný programovací jazyk vyvíjaný firmou Adobe systems pre platformu Flash.

AS 1.0 – prvá verzia programovacieho jazyka ActionScript.

AS 2.0 – druhá verzia programovacieho jazyka ActionScript, pokračovanie AS 1.0.

AS 3.0 – tretia verzia jazyka ActionScript následník jazyka AS 2.0.

ADO.Net – časť programovacích jazykov rodiny .Net, ktorá zabezpečuje spracovávanie dát v databázovej podobe.

API – *Application Programming Interface*, zbierka procedúr funkcií či tried určitej knižnice alebo programu.

ASP – *Active Server Pages*, technológia pre tvorbu webových aplikácií.

ASP.Net – súčasť .Net frameworku, ktorá slúži k tvorbe serverových webových aplikácií, je založená na funkcionalite technológie ASP.

AVM – *ActionScript virtual machine*, je sada počítačových programov a štruktúr, ktoré využívajú modul virtuálneho stroja k spusteniu ďalších počítačových programov a skriptov vytvorených v jazyku AS.

AVM v 2 – *ActionScript virtual machine version 2.0*, je pokračovaním AVM, prináša viaceré novinky ako napríklad spracovanie XML, binárna socketová(viď časť 3.4) komunikácia, spracovanie regulárnych výrazov a mnohé iné.

db(báza dát) – databáza, konkrétne v tejto práci je pod týmto pojmom myslený relačný databázový systém napr. typu Oracle.

ECMAScript – *European Computer Manufacturers Association Script*, štandardizovaná verzia jazyka JavaScript firmou European Computer Manufacturers Association, v užšom slova zmysle je pod týmto pojmom myslená technická špecifikácia tohto jazyka.

hloop – *high level object oriented programming languages*, vysoko úrovňové programovacie jazyky ako sú napr. Java a C++ / C#.

Http – *HyperText transfer protokol*, internetový protokol určený na výmenu hypertextových – html dokumentov v prostredí internetu.

JSP – *JavaServer Pages*, je to technológia určená k tvorbe dynamického webu, založená na jazyku Java.

JVM – *Java virtual machine*, , je sada počítačových programov a štruktúr, ktoré využívajú modul virtuálneho stroja k spusteniu ďalších počítačových programov a skriptov vytvorených v jazyku Java.

ns – *namespace*, t.j. menný priestor, je abstraktný kontajner združujúci spolu súvisiace identifikátory a symboli – vlastnosti(viď časť 2.1).

ODAC – *Oracle Data Access Components*, súbor prostriedkov umožňujúcich integráciu db. Oracle s hloop jazykmi.

OOP – *Object oriented programming*, je súbor princípov a pravidiel určujúci štýl programovania zamariavajúci sa na objekt.

query string – Špeciálna časť URL slúžiaca k prenosu dát.

URL – *Uniform resource locator*, je špeciálny text, ktorý identifikuje internetový zdroj.

XML – *Extensible markup language*, je špeciálny jazyk využívajúci párových značiek k uchovávaniu dát. Využíva sa najmä v prostredí internetu. V užšom slova zmysle pod týmto pojmom rozumieme súbor pravidiel pre tvorbu takýchto dokumentov špecifikovaný v W3C.

XML – RPC : *Remote procedure calls* – vzdialené procedurálne volanie, je protokol využívajúci XML ku kódovaniu svojich volaní a Http prenosový mechanizmus k prenosu po sieti.

Kt – ktorý, ktorá, ktoré

Obsah :

1.0. Action Script v 3.0

1.1. Úvod [1] :

Jazyk ActionScript v 3.0 je objektovo orientovaný programovací jazyk vyvíjaný spoločnosťou Macromedia Inc. dnes Adobe Systems. Tento jazyk je určený primárne k vývoji aplikácií pomocou Macromedia Flash.
ActionScript vychádza zo štandardizovanej verzie jazyka JavaScript nazvanej ECMAScript. Tak ako už názov napovedá je už treťou verziou svojej generácie.
Jazyk ActionScript 3.0 (ďalej už len AS3) je silný nástroj pre vývoj rôznych bohatých internetových aplikácií tzv. RIA aplikácií a aplikácií, ktorých funkčnosť je založená na protokole XML-RPC.
Možnosti jazyka AS3 nie sú obmedzené len na vývoj týchto dvoch typov aplikácií.
Jazyk AS3 je plnohodnotným objektovo orientovaným jazykom a poskytuje vývojárovi takmer všetky výhody štandardných „high level objeckt oriented programing languages“ ako napríklad C# alebo Java.
V skutočnosti práve funkcionalitou prevádzania kódu a jeho syntaxe je veľmi podobný jazyku Java.

1.2. Jazyk ActionScript 3.0 – výhody :

Jazyk AS3 je oproti svojim predchodcom až desať krát rýchlejší a to vďaka využívaniu ActionScript Virtual Machine v2 (povšimnite si podobnosť s Java Virtual Machine).
AS3 vysoko prekonáva skriptovacie schopnosti svojich predchodcov a to vďaka rozsiahlejším API a dokonalejším knižniciam, ktoré poskytujú lepšiu kontrolu objektov na nižších úrovniach.
Jeho účelom je vytváranie vysoko komplexných aplikácií spracovávajúcich rozsiahlu škálu dát a využívajúcich objektovo orientovaný prístup. Zameriava sa na opätovne využiteľné časti kódu.

1.3. ActionScript 3.0 a Objektovo orientované programovanie:

Táto kapitola a nasledujúce kapitoly 1.4. až 1.4.4. vychádzajú z **[2]**.
Vďaka kmeňom jazyka AS ako skriptovacieho jazyka je podpora Objektovo orientovaného programovania(ďalej len OOP) voliteľná.
Táto možnosť ponúka programátorom flexibilitu vo výbere najlepšieho programovacieho prístupu pre projekty rôzneho rozsahu a zložitosti. Pre malé projekty môže postačiť použitie jazyka AS s paradigmatom procedurálneho plánovania. Rozsiahlejšie projekty budú vďaka princípom OOP jednoduchšie na pochopenie, údržbu a budú mať lepšiu škálovateľnosť.

---

[1] Viac informácií o jazyku AS nájdete na : **[1]**

1.4. Aspekty OOP v AS3:

- **1.4.0. Triedy:**

Už od verzie jazyka AS 1.0 mohli programátori využívajúci jazyk AS používať objekty a funkcie pre vytvorenie konštrukcií, ktoré sa podobali triedam.
AS 2.0 pridal formálnu podporu tried pomocou kľúčových slov ako napr. : `class` a `extends`.
AS 3.0 naďalej podporuje využívanie týchto kľúčových slov a zároveň pridáva niektoré rozšírené možnosti, ktoré sme mohli pozorovať u iných „high level" programovacích jazykov, a to napríklad vylepšenie ovládania prístupu k atribútom jednotlivých objektov pomocou nových možností deklarovania atribútu cez kľúčové slová `protected` a `internal`, ktorých funkcionalita je už zrejmá z iných jazykov ako je Java alebo C#.
Ďalej AS 3.0 umožňuje lepšiu kontrolu a ovládanie dedičnosti zabezpečené pomocou možnosti využitia kľúčových slov `final` a `override`, s ktorými sa mohla už väčšina programátorov stretnúť v jazyku Java a C#.
Jazyk AS 3.0 pri práci s triedami využíva mnoho vyhradených slov identických s jazykmi ako sú C++, Java alebo C#.
Aj keď sa môže zdať, že jazyk AS 3.0 kopíruje iné programovacie jazyky, čo sa tried týka, nie je tomu úplne tak. Dokazujú to drobné rozdiely ako napríklad:

V AS 3.0 majú triedy štyri základné kombinovateľné vlastnosti, a to:

- **Dynamic**: umožňuje pridávanie vlastností jednotlivým inštanciám danej v čase spracovania.
- **Final**: oznamuje, že trieda nesmie byť rozšírená inou triedou.
- **Internal(default)**: vlastnosť určujúca, že trieda je viditeľná len vnútri aktuálneho balíka. Táto vlastnosť je priradená k triedám implicitne bez nutnosti zahrnutia do deklarácie. Pokiaľ nechceme aby bola trieda viditeľná len v rámci daného balíka použijeme nasledujúci atribút – public.
- **Public**: trieda je viditeľná kdekoľvek. Zahrnutie tejto vlastnosti do deklarácie triedy ruši implicitne danú vlastnosť – internal.

V AS 3.0 neexistuje vlastnosť tried „abstract", teda AS 3.0 nepodporuje koncept abstraktných tried. Využitie abstraktných tried je nahradené implementáciou rozhraní.

Syntax triedy je zhodný s inými OOP jazykmi.
V tele triedy sa môžu nachádzať: definície premenných, konštánt, metód, iné verejné funkcie a mnohé iné.
AS 3.0 umožňuje aj deklaráciu menných priestorov tzv. „namespacov" v tele triedy.
AS 3.0 umožňuje deklaráciu statického atribútu a triedneho atribútu s rovnakým identifikátorom v rámci jednej triedy.

- **1.4.1. Atribúty vlastností triedy:**

  Pojmom vlastnosť triedy sa rozumie čokoľvek čo môže byť členom triedy, t.j. premenné, konštanty a metódy.
  Jazyk AS 3.0 definuje nasledujúcu množinu atribútov vlastností triedy:
  - **Internal(default)** : atribút je viditeľný vnútri daného balíku.
  - **Private**: atribút je viditeľný len v danej triede, tieto vlastnosti sú nedostupné v čase kompilácie aj v čase spracovania.
  - **Protected**: atribút je viditeľný v danej a odvodených triedach.
  - **Public**: atribút je verejne prístupný.
  - **Static**: určuje, že atribút patrí danej triede skôr ako k danej inštancií – objektu.
  - **UserDefinedNamespace**: názov vlastného „namespace“ definovaného užívateľom, alternatíva niektorého z predchádzajúcich, takýto atribút je dostupný v rámci daného „UserDefinedNamespace“.
  - **Static**: špecifický atribút, ktorý sa dá použiť s vlastnosťami deklarovanými pomocou kľúčových slov: `var, const` alebo `function`. Určuje že daný atribút sa vzťahuje na triedu v ktorej sa nachádza nie len na danú inštanciu triedy.
    Kód mimo danej triedy volá takýto atribút prostredníctvom identifikátoru triedy, nie inštancie. Statické vlastnosti nie sú zdedené podtriedami, ale dané podtriedy ich môžu volať bez nutnosti identifikácie prostredníctvom triedneho identifikátoru, teda priamo.

- **1.4.2. Metódy:**

  Metódy sú funkcie, ktoré sú súčasťou kódu triedy, teda nachádzajú sa v tele triedy. Sú deklarované pomocou vyhradeného slova `function` a ľubovoľne zvolenej vlastností triedy, z vlastností uvedených v 1.4.1, určujúcej prístup k danej metóde. V AS 3.0 ostatne aj v iných programovacích jazykoch rozlišujeme niekoľko základných typov metód:

  - **Metódy konštruktoru**: Sú metódy nesúce rovnaký identifikátor ako trieda v ktorej sa nachádzajú. Metódy konštruktoru môžu byť iba verejné použitie vyhradeného slova `public` v AS 3.0 nie je nutné.
    Konštruktory nemajú žiadnu návratovú hodnotu.
    Kód metódy konštruktora sa vykoná vždy pri vytvorení novej inštancie triedy vyhradeným slovom `new`. V prípade, že nevytvoríme žiaden konštruktor v AS 3.0 implicitne dôjde k vytvoreniu prázdneho konštruktoru.
    Pri dedičnosti sa môžeme explicitne odkázať na konštruktor predka pomocou volania `super()`, pokiaľ takéto volanie nevykonáme je explicitne vykonané pred vykonaním prvého príkazu konštruktoru potomka. Taktiež sa môžeme odkazovať na predka prostredníctvom predpony `super`. Pre správnu funkčnosť odkazovania na predka sa doporučuje predtým explicitne zavolať jeho inicializáciu prostredníctvom `super()`.

- **Statické metódy:** Tiež nazývané metódy triedy sú metódy, ktoré priamo neovplyvňujú členské vlastnosti nazývane „member variables“ objektu danej triedy. Tieto metódy sú volané prostredníctvom identifikátoru triedy, nie identifikátoru objektu a sú dôležité z hľadiska správneho zapúzdrenia.
  V tele takýchto metód nemožno použiť identifikátory `this` alebo `super`, pretože takáto metóda sa viaže k danej triede, nie jej inštanciám. Na rozdiel od iných OOP jazykov statické metódy nie sú v AS 3.0 dediteľné, ale potomkovia sa k nim môžu odvolávať bez nutnosti použitia identifikátoru triedy.
- **Členské metódy:** bežné metódy vzťahujúce sa ku konkrétnej inštancií prostredníctvom ktorej sú volané. V takýchto metódach môžeme využívať referencie `this`, ktorá odkazuje na danú inštanciu resp. objekt.
  Tieto metódy sú deklarované bez vyhradeného slova `static`, sú teda nestatické, ale takáto metóda môže pracovať so statickými premennými.
  Dedičnosť metód môžeme v jazyku AS 3.0 ovládať rovnako ako v jazyku Java prostredníctvom kľúčových slov `override` a `final`.
  `Override` je atribút, ktorý môžeme využiť pre novú definíciu metódy. Naproti tomu `final` je atribút, ktorý značí, že metóda nemôže byť potomkom nahradená.
- **Metódy prístupu:** Tiež nazývané getset metódy. Sú pomocné metódy slúžiace k správnemu zapúzdreniu privátnych vlastností objektu. V podstate sa jedná o dve verejné metódy s atribútmi `get` a `set` indentifikované rovnakým identifikátorom. Vo finálnom výsledku teda pracujeme s privátnym atribútom mimo danej triedy prostredníctvom konkrétnej inštancie a vďaka rovnakému identifikátoru metód, akoby jednou metódou, aj keď sú dve. Kompilátor už na základe použitia určí či danú vlastnosť nastavujeme – set alebo len získavame – get.
  Tieto metódy sú tiež dôležité z pohľadu dedičnosti. Vlastnosti deklarované prostredníctvom príkazu `var` nie je možné v podradenej triede potlačiť. Avšak vlastnosti, ktoré sme získali prostredníctvom využívania metód prístupu môžeme prostredníctvom príkazu `override` potlačiť.
- **Metódy väzby:** Sú metódy vybrané zo svojej inštancie. Sú to metódy, ktoré sú vrátené inou metódou alebo, tak ako je tomu u mechanizmu spracovania udalostí, metódy odoslané ako parameter. To, že je metóda vybraná zo svojho prirodzeného lexikálneho prostredia má za následok, že kľúčové slovo `this` nie vždy odkazuje na pôvodný objekt ktorý danú metódu implementuje.
  AS 3.0 preto pri odoslaní metód ako parameter vždy automaticky vytvára metódu väzby, čo zabezpečuje, že vyhradené slovo `this` vždy odkazuje na objekt alebo triedu, v ktorej bola daná metóda vytvorená.
  Vytváranie metód väzby je najviac zrejmé pri spracovaní udalostí, pretože metóda `addEventListener()` vyžaduje odoslanie metódy ako parameter.

- **1.4.3. Rozhranie:**

  Viacnásobná dedičnosť a abstrakcia je v AS 3.0 rovnako ako je tomu u jazykov Java a C# nahradená implementáciou rozhraní.
  Syntax a spôsob použitia rozhraní je v AS 3.0 plne zhodný s Javou.
  Pár pravidiel pre prácu s rozhraniami:
  - Deklaruje sa prostredníctvom vyhradeného slova `interface`.
  - Triedy môžu implementovať nula až ľubovoľne veľa rozhraní prostredníctvom vyhradeného slova `implements`. V prípade implementácie niekoľkých rozhraní sú jednotlivé rozhrania v deklarácií implementácie oddelené čiarkou.
  - Atribúty prístupu pre rozhrania môžu byť len `public` alebo `internal`.
  - Deklarácia rozhrania sa nesmie nachádzať v triede alebo inom rozhraní.
  - Jedno rozhranie môže implementovať teda rozširovať iné rozhrania.
  - Názov rozhrania by sa podľa dohody mal začínať písmenom „I“, ale všeobecne sa dá použiť akýkoľvek platný identifikátor.
  - Rozhranie nesmie byť konkretizované, preto jeho telo obsahuje len definície deklarácií, jedná sa teda o určitý abstraktný predpis.
  - Metódy nachádzajúce sa v rozhraní nemajú žiaden atribút prístupu, ku konkretizácií dochádza až v danej triede implementujúcej rozhranie.
  - Rozhranie neobsahuje deklarácie premenných a konštánt, ale môže obsahovať definície metód získania prístupu.
  - Metódy môžu a nemusia mať nastavený predvolený počet parametrov.

- **1.4.4.Vymenované triedy:** Jazyk AS 3.0 nepodporuje vymenované dátové typy tzv. enumy.
  Tieto sa v AS 3.0 nahrádzajú špeciálnym tipom tried, ktoré vo svojom tele obsahujú staticky definované konštanty, ktoré reprezentujú jednotlivé položky vymenovaných dát.
  Podľa dohody sa všetky takéto triedy označujú atribútom `final`, pretože ich netreba ďalej rozširovať.
  Keďže takéto triedy obsahujú len statické triedy, tieto triedy nevytvárajú inštancie.
  K hodnotám vymenovanej triedy preto pristupujeme tzv. triedneho objektu, t.j. identifikátor danej triedy nasledovaný bodkou a identifikátorom danej položky vymenovanej triedy.

Z predchádzajúcich informácií o AS 3.0 vyplýva, že sa jedná o programovací jazyk, ktorý má širokú škálu možností ako sa vyrovnať s bežnými problémami OOP a poskytuje vývojárovi pestrú paletu programátorských nástrojov, ktoré umožňujú zvládnutie bežných princípov OOP ako sú zapúzdrenie teda encapsulácia, polymorfizmus a dedičnosť.

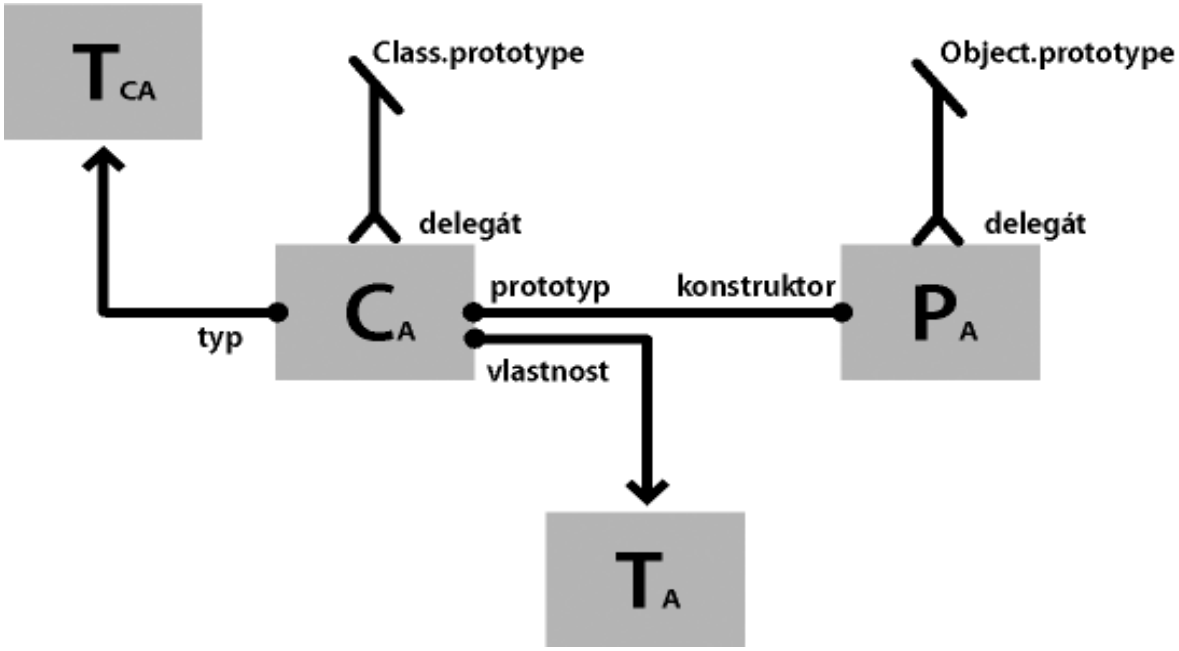
2.0. ActionScript 3.0 dedičnosť a objekty v ňom

Táto kapitola vychádza z **[3]**.
Dedičnosť v AS 3.0 a v štandardných programovacích jazykoch ako je Java, C++ a iných na jazyku C založených jazykoch sa mierne líši. AS 3.0 na rozdiel od týchto jazykov rozlišuje niekoľko typov objektov:

- **2.0.1. Objekt triedy:**

  Bežné paradigma programovania orientovaného na objekt, kt. je spojované s jazykmi Java a C++, používa triedy pre definíciu tried objektov. Inými slovami trieda je len akási platforma pre tvorbu objektov, s ktorými sa ďalej pracuje.
  Iným typom programovacích jazykov sú jazyky, ktoré používajú triedy pre konštrukciu inštancií typu dáta, ktoré trieda definuje.
  Jazyk AS 3.0 využíva triedy k obom týmto spôsobom.
  AS 3.0 vytvára pre každú definíciu triedy zvláštny objekt triedy, ktorý umožňuje zdieľanie chovania i stavu.
  Základ jazyka založeného na prototype pridáva ďalšie zaujímavé a rozširujúce možnosti. Štandardne tento objekt využívame k práci so statickými premennými a statickými metódami, ale nemusí tomu byť tak.

Následný diagram ponúka lepší náhľad na štruktúru objektu triedy, ktorý predstavuje jednoduchú triedu pomenovanú A, definovanú príkazom `class A {}`:



Každý obdĺžnik diagramu predstavuje objekt. Každý objekt tohto diagramu má charakter dolného indexu A, kt. predstavuje to, že náleží do triedy A. Objekt CA obsahuje odkazy na niekoľko dôležitých objektov. Objekt vlastnosti inštancie TA uchováva vlastnosti inštancie, kt. sú definované v rámci definície triedy. Objekt vlastnosti triedy TCA predstavuje interný typ triedy, uchovávajúci statické vlastnosti definované danou triedou. Objekt prototypu PA vždy odkazuje na objekt triedy, ku ktorému bol pôvodne pripojený pomocou vlastnosti `constructor`, to sa využíva pri tvorbe väzobných metód(viď metódy väzby str. 10).

- **2.0.2. Objekt vlastnosti:**

  Objekt vlastnosti je novinkou jazyka AS 3.0 a v predchádzajúcich verziách sa nevyskytoval.
  Bol implementovaný s ohľadom na výkon. V starších verziách jazyka AS 3.0 mohlo byť vyhľadávanie názvu, kde Flash Player prechádza reťazec prototypu, časovo náročným procesom. V AS 3.0 je vyhľadávanie názvov ďaleko účinnejšie a menej časovo náročné, pretože zdedené vlastnosti sú skopírované z nadradených tried do tried objektov vlastností podtried.
  Objekt vlastnosti nie je priamo dostupný kódom, ale jeho prítomnosť je znateľná na lepšom výkone a využití pamäti. Objekty vlastností poskytujú AVM2 detailné informácie o rozvrhnutí a obsahu triedy. Vďaka informáciám uloženým v tomto objekte môže AVM2 výrazne skrátiť dobu prevedenia, pretože môže generovať priame pokyny zariadenia pre prístup k vlastnostiam alebo môže vyvolať metódy priamo bez časovo náročného vyhľadávania názvu.
  Vďaka objektu vlastností môže byť odtlačok pamäti objektu výrazne menší než podobný objekt v predchádzajúcich verziách jazyka AS. Ak je trieda zapečatená t.j. nie je deklarovaná ako dynamická, inštancia triedy nepotrebuje tabuľku rušenia pre dynamicky pridávané vlastnosti. Môže len prestavovať ukazovadlo na objekty vlastností a niektoré bloky pre pevné vlastnosti definované v danej triede. Následkom toho môže objekt, kt. by v AS 2.0 mal pamäťový odtlačok 100bytov, mať odtlačok veľkosti len 20 bytov v AS 3.0.

- **2.0.3. Objekt prototypu:**

  Každý objekt triedy jazyka AS má vlastnosť pomenovanú `prototype`, ktorá je odkazom na objekt prototypu triedy. Objekt prototypu je odkazom koreňov jazyka AS ako jazyka založeného na prototypoch.
  Prototypová dedičnosť bola jediným typom dedičnosti v predchádzajúcich verziách ActionScriptu.
  V jazyku AS 1.0 neexistovala definícia triedy, pre vytvorenie triedy sa vytvárala funkcia konštruktoru danej triedy.
  V tomto jazyku boli funkcie skutočné objekty, nie len abstraktné definície.
  Funkcia konštruktoru, ktorá sa vytvorila fungovala ako prototypický objekt pre inštancie danej triedy.
  Následný kód vytvára triedu „shape“ a definuje jej jednu vlastnosť.

```
function Shape() {} // base class
// Create a property named visible.
Shape.prototype.visible = true;
```

  Vytvorenie inštancií potom prebiehalo bežným spôsobom pomocou volania metódy `new()` a využitia prototypického konštruktora.

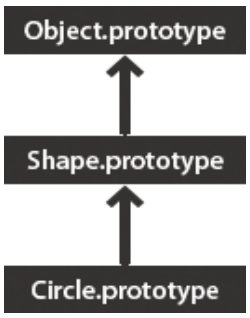
```
myShape = new Shape();
```

  Rovnako ako konštruktor triedy `Shape()` slúži ako prototyp pre inštancie triedy `Shape()`, tak môže slúžiť ako prototyp pre podtriedy tejto triedy. Deklarovanie dedičnosti by teda vypadalo takto:

```
function Circle(id, radius)  // child class
{
        this.id = id;
        this.radius = radius;
}
// Make Circle a subclass of Shape.
Circle.prototype = new Shape();
```

Dedičnosť teda nebola ničím iným ako postupným reťazcom prototypov jednotlivých tried implementujúcich dedičnosť.
Tento prístup bol veľmi neefektívny, pretože pri hľadaní určitej vlastnosti bol prehľadaný stávajúci prototyp a následne prototyp tohto prototypu až po základnú triedu.
Základná trieda ukončujúca každý reťazec dedičnosti bola trieda `Object`. Táto trieda deklarovala statickú vlastnosť `Object.prototype`, ktorá ukazuje na základný objekt prototypu pre všetky objekty vytvorené v jazyku ActionScript 1.0.

Následná schéma zachytáva prototypický reťazec dedičnosti pre predchádzajúci príklad:



V jazyku AS 3.0 je vlastnosť prototype určená len k čítaniu, čo znamená, že nemôže byť upravená a odkazovať tak na rôzne objekty na rozdiel od predchádzajúcich verzií jazyka AS. Aj keď je vlastnosť `prototype` určená len na čítanie, objekt prototypu, na ktorý táto vlastnosť odkazuje, nie je. Inými slovami, k objektu prototypu je možné pridať nové vlastností a odkázať sa naň resp. získať tento objekt môžeme práve prostredníctvom vlastnosti `prototype`. Vlastnosti pridané k objektu prototypu sú potom zdieľané medzi všetkými inštanciami danej triedy a reťazca dedičnosti.

Reťazec prototypov bol jediným mechanizmom dedičnosti v predchádzajúcich verziách ActionScriptu, v AS 3.0 má len sekundárnu úlohu. Primárnym mechanizmom dedičnosti je dedičnosť pevnej vlastnosti. Takáto dedičnosť je spracovávaná interne objektom vlastnosti. Pod pojmom pevná vlastnosť sa rozumie premenná alebo metóda definovaná ako súčasť definície triedy. Dedičnosť pevnej vlastnosti sa preto nazýva aj dedičnosť triedy a spája sa s výrazmi zavedenými jazykom AS 2.0, za účelom obmedzenia pomalej prototypovej dedičnosti, ako sú napríklad `class`, `extends` a `override`. Inšpirácia pre tento typ dedičnosti sa našla v jazyku Java.

Reťazec prototypu poskytuje alternatívny mechanizmus dedičnosti, ktorý je dynamickejší ako dedičnosť pevnej vlastnosti. Vlastnosti objektu prototypu triedy môžeme pridať nielen ako súčasť definície triedy, ale taktiež v čase spracovania práve prostredníctvom triedy vlastností `prototype` objektu. Aby bolo toto možné je nutné danú triedu deklarovať s modifikátorom prístupu dynamic.
Dobrým príkladom triedy s niekoľkými vlastnosťami pripojenými k objektu prototypu je základná trieda `Object`. Všeobecné známe metódy tejto triedy `toString()` a `valueOf()` sú vlastne funkcie priradené k vlastnostiam objektu prototypu triedy `Object`. Nasledujúci príklad ukazuje ako by mohla ich implementácia teoreticky vypadať(skutočná implementácia sa mierne líši z dôvodov detailov implementácie).

Implementácia metód `toString()` a `valueOf`() prostredníctvom využitia objektu `prototype.`

```
public dynamic class Object
      {
            prototype.toString = function()
            {
                  // statements
            };
            prototype.valueOf = function()
            { // statements
            };
      }
```

Ako už bolo spomínané, je možné pripojiť vlastnosť k objektu prototypu triedy mimo definíciu triedy. Napr. metódu `toString`() by sme mohli deklarovať aj takto:

```
Object.prototype.toString = function()
{
      // statements
};
```

Na rozdiel od dedičnosti pevnej vlastnosti dedičnosť prototypu nevyžaduje kľúčové slovo `override`.
Napr. ak chceme znova definovať metódu `valueOf()` v podtriede triedy `Object`.
Máme tri možnosti:

1. Definovať metódu na objekte prototypu členskej triedy vnútri definície triedy. Nasledujúci kód vytvorí podtriedu objektu pomenovanú „MyClass“ a znova definuje metódu `valueOf()` na objekte prototypu triedy `MyClass` ako súčasť definície tejto triedy. Keďže každá trieda dedí z triedy `Object,` nie je potrebné použiť definíciu dedenia prostredníctvom `extends`.

   ```
   dynamic class MyClass
   {
         prototype.valueOf = function()
         {
               return "Instance of MyClass";
         };
   }
   ```

2. Definovať metódu `valueOf()` na objekte prototypu triedy `MyClass` mimo definíciu tejto triedy:

   ```
   MyClass.prototype.valueOf = function()
   {
         return "Instance of MyClass";
   };
   ```

3. Definovať pevnú vlastnosť danej triedy pomenovanú `valueOf()`. Toto kombinuje oba typy dedičnosti, t.j. dedičnosť pevnej vlastnosti aj prototypu. Akákoľvek podtrieda triedy `MyClass` musí použiť na predefinovanie takejto metódy príkazu `override`.

   ```
   class MyClass
   {
         function valueOf():String
         {
               return "Instance of MyClass";
         }
   }
   ```

2.1. Menné priestory – namespaces v AS 3.0

Existencia dvoch oddelených mechanizmov dedičnosti(pevnej vlastnosti a prototypu), vytvára otázku kompatibility s ohľadom na vlastnosti a metódy tried jadra jazyku AS 3.0. Zlučiteľnosť so špecifikáciou jazyka ECMAScript, na ktorých je jazyk ActionScript založený vyžaduje použitie dedičnosti prototypov, čo v praxi a kóde znamená, že vlastnosti a metódy tried jadra sú definované na objekte prototypu danej triedy. Na druhú stranu zlučiteľnosť s jazykom ActionScript 3.0 vyžaduje použitie dedičnosti pevnej vlastnosti, čo znamená, že vlastnosti a metódy tried jadra sú definované v definícií triedy za pomoci príkazov ako napr. `const`, `var` a `function`. Navyše požitie dedičnosti pevných vlastností miesto verzií prototypov môže viesť k výraznému nárastu výkonu počas spracovania.

Jazyk ActionScript 3.0 tento problém rieši použitím oboch týchto dedičností pre triedy jadra. Každá trieda jadra obsahuje dve množiny vlastností a metód. Jedna množina je definovaná na objekte prototypu a druhá je definovaná pomocou pevných vlastností a menného priestoru AS3.

Práve menný priestor AS3 poskytuje výhodný mechanizmus pre voľbu medzi týmito dvoma množinami vlastností a metód. Jednoducho ak vývojár nepoužíva menný priestor AS 3.0, inštancie tried jadra definovaných v AS dedia vlastnosti a metódy definované na objekte prototypu triedy jadra. Ak sa vývojár rozhodne použiť menný priestor AS3, inštancie tried jadra dedia vždy pevne definované vlastnosti z tried jadra, pretože pevné vlastnosti sú v AS 3.0 uprednostňované pred vlastnosťami prototypov.

Vývojár môže selektívne využívať výhod menného priestoru AS3 a to selekciou vlastností danej triedy s týmto menným priestorom. Nasledujúci kód ukazuje volanie metódy `Array.pop()` z namespace(ďalej len ns) AS3.

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

Taktiež môže použiť príkaz `use namespace` a otvoriť tak menný priestor AS3 pre všetky definície v rámci daného bloku kódu. Nasledujúci kód ukazuje sprístupnenie ns AS3 pre blok kódu a volanie metód `pop()` i `push()` z tohto priestoru.

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

AS 3.0 taktiež poskytuje voľby kompilátoru pre každú sadu vlastností, takže ns AS3 sa použie na celý projekt. Pre otvorenie ns AS3 pre celý program sa nastaví voľba kompilátoru `-as3`(pevná dedičnosť) na hodnotu `true` a voľba `-es`(dedičnosť prototypu) na `false`. Pre opačnú voľbu stači prehodiť `true` a `false` pre oba indikátory.

Predvolené nastavenie kompilátoru pre aplikácie Adobe Flex Builder 3 a Adobe Flash CS4 Professional sú `-as3 = true` a `-es = false`.

Voľba ns má výrazný vplyv v okamihu, keď sa vývojár snaží rozšíriť – upraviť niektorú z tried jadra.
Z predchádzajúcich informácií o dedičnosti a objekte vlastností a prototypu vyplýva, že pre rozšírenie – úpravu metód z ns AS3(ktoré sú pevne definované) musí vývojár daný ns využívať - `use` a využiť kľúčového slova `override`. V prípade rozšírenia metódy prostredníctvom vlastnosti prototypu by nemal vývojár používať ns AS3 a ani príkaz `override`.

Obidva spôsoby dedičnosti a hlavne flexibilná možnosť prepínania medzi nimi ponúka programátorovi možnosť ako optimalizovať svoju aplikáciu čo najlepšie pre dané potreby a tým zlepšiť výkon svojej aplikácie.

3.0. Možnosti prepojenia báze dát s ActionScript 3.0

Síce je AS3.0 veľmi silný OOP jazyk, ale stále sa jedná len o jazyk, bežiaci v klientskom prostredí prehliadača. Z toho dôvodu nemá možnosť pripojenia sa a spolupráce s rozličnými databázami. Pokiaľ chceme túto funkcionalitu dosiahnuť, je nutné využiť určitého prostredníka, ktorý bude slúžiť ako most k báze dát, takýmto prostredníkom je „server-side" technológia schopná spolupráce s databázou a zároveň schopná komunikácie s „client-side" prostredím reprezentovaným jazykom AS 3.0.

Existujú tri základné technológie, ktoré zabezpečujú spojenie báze dát a ActionScript 3.0:

- **ASP .NET**: *Active ServerPpages*, technológia vyvinutá spoločnosťou Microsoft, ktorá môže využívať všetky výhody, ktoré ponúkajú jazyky ako C# a Visual Basic.
- **JSP**: *Java Servlet Pages*, server-side technológia založená na jazyku Java.
- **PhP**:*Personal Home Page* – hypertextový preprocesor, skriptovací jazyk určený k tvorbe dynamického webu s výkonov logikou na strane serveru fungujúci na princípe komunikácie client-server.

Každá z týchto technológií má svoje silné aj slabšie stránky, ktorým sa budem ďalej podrobnejšie venovať.

3.1. Integrácia AS 3.0 a „server-side" technológií [2]

Základným komunikačným prostriedkom jazyka AS 3.0 sú triedy `URLLoader`, `URLRequest` a `URLVariables`.
Triedy `URLLoader` a `URLVariables` sú triedy využívané k spracovaniu externých dát.
Trieda `URLLoader` slúži k zisku dát z určitého URL a získané dáta sú buď textové alebo binárne, taktiež môže táto trieda slúžiť k načítaniu URL-kódovaných premenných, t.j. premenných uložených do textu ako dvojica identifikátor = hodnota a oddelených znakom ‚&'. Typicky sa premenné v tomto formáte posielajú ako tzv. „queryString".
V AS 3.0 sa s takýmito premennými pracuje prostredníctvom triedy `URLVariables`, v podstate sa jedná o slovník uchovávajúci hodnoty jednotlivých premenných, ktorý obsahuje metódy slúžiace k zjednodušeniu získavania a vkladania informácií.
Trieda `URLLoader` využíva výhod vylepšeného modelu spracovania udalostí, ktorý ponúka jazyk AS 3.0, práve vďaka tomu môžeme na tejto triede pracovať s udalosťami ako sú: `complete`, `httpStatus`, `ioError`, `open`, `progress`, a `securityError`.
Najmä udalosť `complete`, ktorá signalizuje, že všetky dáta boli načítané do inštancie triedy `URLLoader` v poriadku, je veľmi často využívaným zlepšením.
Zachytenie udalosti ako `ioError`, `securityError` výrazným spôsobom uľahčuje spracovanie výnimiek.
Dáta, ktoré sú načítavane do triedy `URLLoader` nie sú dostupné, pokiaľ nie sú všetky kompletne načítané – udalosť `complete`.
Pokiaľ je posielaná väčšia množina dát je možné načítavanie dát monitorovať prostredníctvom udalosti `flash.events.ProgressEvent.PROGRESS`. Konkrétne sa dá prostredníctvom tejto udalosti zistiť koľko bytov bolo odoslaných a koľko bytov majú odosielané dáta celkovo.

---

[2] Oficiálne texty a následne dokumentácia k v texte uvedeným triedam **[4]**.

Po načítaní sú dáta prekódované z kódovania UTF 8 alebo UTF 16 do typu `String`.

Najjednoduchší spôsob prenosu dát medzi AS 3.0 a „server-side“ technológiami je posielanie premenných za pomocí http metód GET a POST vo formáte „query stringu“.

Jednoduchý príklad kódu umožňujúci integráciu ASP.Net a AS 3.0:

3.1.0. AS 3.0 : Metóda get a komunikácia prostredníctvom query stringu [3]

```
import flash.events.Event;
import flash.net.*;
      function send ()
      {
            var myRequest:URLRequest = new URLRequest ();
            var myVariables:URLVariables = new URLVariables();
            var myLoader:URLLoader = new URLLoader();

            myRequest.url = "http://localhost:55470/Default.aspx";
            //URL mojej ASP stranky

            myVariables.nick = "NickKOdoslaniu";
            myRequest.method = URLRequestMethod.GET;
            myRequest.data = myVariables;
            myLoader.dataFormat = URLLoaderDataFormat.VARIABLES;

            function sendComplete(e:Event):void
            {
                  //trace('complete');
                  if(e.target.data.response==null)
                  {
                        trace('server neposlal ziadnu odpoved');
                  }
                  else
                  {
                        trace('server odpovedal takto' +
                        e.target.data.response);
                  }
            }
            myLoader.addEventListener(Event.COMPLETE, sendComplete);
            myLoader.load(myRequest);
        }
```

Funkcia odošle na server dáta, v tomto prípade nejaký nick a zachytáva odpoveď serveru – response, ktorú následne vypíše.
Dáta sú odoslané prostredníctvom queryStringu čo zabezpečuje trieda `URLRequest`, ktorej treba nastaviť tri parametre a to:

1. `URLRequest.url` : URL adresa serveru, s kt. sa chystáme komunikovať.
2. `URLRequest.method` : vyberáme z dvoch možností a to `URLRequestMethod.GET` a `URLRequestMethod.POST`.

[3] Viac informácií a príkladov nájdete na **[5]** a **[6]**

Pozor pri použití metódy POST je potrebné správne nastavenie servera ako napr. nastavenie firewalu a pod., pretože prichádzajúce dáta prostredníctvom metódy `Post` môžu byť serverom blokované, záleží od nastavenia.
Preto, pokiaľ sa jedná o verejné dáta je možné k odoslaniu dát na server použiť metódy Get tak ako som to spravil v predchádzajúcom kóde. Takéto dáta sú však voľne viditeľné každému kto monitoruje sieťový prenos, napr. prostredníctvom programu wireshark a pod.
Pokiaľ sú odosielané citlivé dáta je výhodnejšie využiť metódy Post a správne nastaviť server. Ale aj tak sú tieto dáta pre rôznych útočníkov ľahko zhliadnuteľné.
Z toho dôvodu je najistejším riešením použitie zabezpečeného spojenia `HTTPS`.

3. `URLRequest.data` : dáta , ktoré majú byť odoslané.

Ďalej treba triede `URLLoader` povedať v akom formáte bude dáta príjmať resp. odosielať. To dosiahneme prostredníctvom nastavenia parametru `URLLoader.dataFormat`.
Opäť sa nám ponúkajú tri možnosti a to: `URLLoaderDataFormat.BINARY`, `URLLoaderDataFormat.TEXT` a `URLLoaderDataFormat.VARIABLES`. Už z názvov je badateľné, že binary bude slúžiť k odoslaniu binárnych dát t.j dát vo formáte `byteArray`, text pre odoslanie textu, tu patrí aj `XML`, rozdiel bude len v spracovaní a variables t.j. odoslanie slovníka identifikátor = hodnota.
O vytvorenie queryStringu v podobe, kt. pozná C# trieda QueryString sa stará trieda `URLVariables`, kt. dáta do nej vložené transformuje na textový reťazec tvaru `mojeURL?identifikátor1=hodnota1&identifikátor2=hodnota2...`
Pozor táto trieda takéto reťazce formuje automaticky, ale pri odpovedaní často formuje reťazce manuálne programátor, preto si treba dať pozor na znak `&`, ktorý slúži ako oddeľovač. Pre správne spracovanie touto triedou musíme za každou dvojicou identifikátor premenná tento znak vložiť, aj keď posielame jedinú premennú. Pokiaľ chceme tento znak vložiť do reťazce bez funkcionality oddeľovača je to potrebné spraviť prostredníctvom jeho `ASCII` kódového označenia.
Vnorená funkcia `sendComplete` je funkcia slúžiaca na obsluhu udalostí kompletného načítania alebo odoslania dát. Povšimnite si v nej použitia vyhradeného slova `target`, ktoré je referenciou na vlastnosť na ktorú je táto obslužná funkcia volaná. V našom prípade je to inštancia triedy `URLLoader`.
Pripojenie udalostí na inštanciu určitej triedy realizujeme prostredníctvom kódu nasledovného formátu: `myObject.addEventListener(Event.typUdalostíKtorúSpracovávame, menoMojejObslužnejMetódy);`
Napokon je potreba na inštanciu triedy `URLLoader` zavolať metódu `load`, ktorá zabezpečí priebeh načítania alebo odoslania dát. Táto metóda má viacero preťažení, buď jej ako parameter odovzdáme inštanciu triedy `URLRequest`, tak ako som to spravil v predchádzajúcom kóde, alebo jej všetky vlastnosti, ktoré táto trieda zastrešuje nastavíme priamo parametrom.

Tento princíp komunikácie zo strany AS 3.0 je pre všetky „server-side“ technológie rovnaký.
Teda kód v AS 3.0 bude rovnaký či už komunikujeme s JSP, ASP alebo PhP, dokonca je tento princíp využiteľný aj na komunikáciu s JavaScriptom a naň nadvezujúcim Ajaxom.

3.1.1. Spracovanie dát na strane servera a odoslanie odpovede späť.

Príklady prevzatia dát od AS 3.0 serverovými technológiami a následná odpoveď serveru späť. Server prijme od AS 3.0 premennú s identifikátorom nick a jej hodnotu odošle späť pod identifikátorom `response`:

3.1.1.0. ASP.Net :

```
<%@ Page Title="My Test Localhost page on URL:
http://localhost:55470/Default.aspx " Language="C#"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<script runat="server">
    public void Page_Load(object sender, EventArgs e)
    {
     if (Request.QueryString["nick"]!= null)
       {
           Response.Write(Request.QueryString["nick"] + "&");
       }
       else Response.Write("response=nonexistingUser&");
    }
</script>
```

Metóda `Page_Load` je obslužnou metódou udalostí načítania danej stránky, kt. sa spustí vždy, keď je daná stránka volaná, zo strany AS 3.0 sa o to stará `URLLoader.load(URLRequest)`.
V C# sú dotazy na danú stránku zastrešené triedou `Request`. Trieda `QueryString` spracováva string vo formáte QueryStringu, v tomto prípade je získany z inštancie triedy `Request`, ďalej sa s ním pracuje ako zo slovníkom `QueryString["identifikátor"]`. O odpoveď serveru sa stará trieda `Response`, konkrétne do nej dáta dostaneme volaním metódy `Write("moja odpoveď")`.

3.1.1.1. JSP : pre funkčnosť treba v kóde pre AS 3.0 treba zmeniť cieľové URL na : `http://host/servlet/packageKdeJeTentoKodUmiestneny.MyAsJspConnection`

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class myAsJspConnection extends HttpServlet
     {
          private static final String CONTENT_TYPE = "text/html;
          charset=windows-1252";
          public void init(ServletConfig config) throws
          ServletException
          {
                    super.init(config);

          }
```

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) throw
      ServletException, IOException
      {
            response.setContentType(CONTENT_TYPE);
            PrintWriter out = response.getWriter();
            out.write("response="+request.getParameter("nick") +
            "&");
            out.close();
      }


public void doPost(HttpServletRequest request,
HttpServletResponse response) throws
      ServletException, IOException
      {
            response.setContentType(CONTENT_TYPE);
            PrintWriter out = response.getWriter();
      }
}
```

Toto je kostra jednoduchého servletu, podrobnejšie nájdete na **[7]** a **[8]**.
Do štandardnej kostry servletu som pridal atribút, `CONTENT_TYPE`, ktorý určuje formát spracovávaného dotazu a odpovedi. Metóda doPost ostáva v nezmenenej podobe, kedže spracovaváme len metódu `GET`.
Metóda `doGet`, zachytáva prichádzajúci dotaz z AS, rovnako ako v C# je požiadavka uložená v inštancií triedy `request`, ku konkrétnym hodnotám tejto požiadavky sa dostaneme cez `request.getParameter("identifikátor")`, ďalej je potreba nastaviť formát odpovedi, urobíme tak cez `response.setContentType(format)`.
Potom treba vytvoriť datový stream, ktorý sa postará o odoslanie dát: `PrintWriter out`, priradíme mu `response.getWriter`(), čo je datový `outputstream` odpovede. Potom už len odošleme dáta prostredníctvom `PrintWriteru` a metódy `write` – `PrintWriter.write("response=moja odpoved" + "&")`. Napokon treba stream zatvorit `PrintWriter.close()`.

3.1.1.2. PhP [4] : pre funkčnosť treba v kóde pre AS 3.0 zmeniť cieľové URL na : `http://localhost/MyAsPhpConnection.php`

```
<?php

$response = $_GET["nick"];

echo $response;

?>
```

V pripade požitia metódy post je potreba zmeniť metódu `$_GET["identifikátor"]` na `$_POST["identifikátor"]`.
`$response` - vytvorenie pomocnej premennej, priradíme jej hodnotu `$_GET["nick"]`, čo je hodnota nicku z query stringu, potom odpoveď odošleme naspäť `echo $response`.

---

[4] Ďalšie príklady nájdete na **[9]**.

3.1.1.3. JavaScript : pre funkčnosť treba v kóde AS 3.0 zmeniť cieľové URL na : `file:///DiskDrive:/cestaKHtmlSuboruDoKtVlozimNasledKod/MyAsJsConnection.html`

Obsah súboru `MyAsJsConnection.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Some title</title>
    <meta name="keywords" content="" />
    <meta name="description" content="" />
    <meta http-equiv="content-type" content="text/html; charset=utf-
     8"/>
    <meta http-equiv="content-language" content="" />
    <link href="style.css" rel="stylesheet" type="text/css" />
<script type="text/javascript">
function getQuerystring(key, default_)
{
     if (default_ == null) default_ = "";
     key = key.replace(/[\[]/, "\\\[").replace(/[\]]/, "\\\]");
     var regex = new RegExp("[\\?&]" + key + "=([^&#]*)");
     var qs = regex.exec(window.location.href);
     if (qs == null)
     {
          return ""; //default_;
     }
     else
     {
          return qs[1];
     }
}
function getResponse()
{
     if (getQuerystring("nick") == null)
     {
          location.href = "MyAsJsConnection.html" +
          "?response=noDataSended";
     }
     else
     location.href="MyAsJsConnection.html"+"?response="+getQuerystri
     ng("nick");
}
</script>
</head>
<body onload="return getResponse()">
</body>
</html>
```

Funkcia `getQuerystring` je funkcia, ktorá na základe zadanej hodnoty v parametri key prostredníctvom regulárnych výrazov prehľadá `query` string a ak nájde zhodu s daným kľúčom vráti jeho hodnotu. Funkcia `getResponse()` využije metódy `getQuerystring` a na základe návratovej hodnoty tejto metódy vytvorí odpoveď, a to buď prázdnu alebo odošle dané dáta späť.

3.2. Limity query stringu a problém s odosielaním väčších a štruktúrovaných dát [5]

Predchádzajúci spôsob prepojenia AS 3.0 a serverových technológií prináša niekoľko problémov :

- Limitovaná veľkosť query stringu – síce samotná veľkosť URL nie je nijak obmedzená, webové prehliadače neakceptujú URL, ktoré majú viac ako 2083 znakov. Pokiaľ bude táto limita prekročená dostaneme chybu 414 `Request-URI Too Long.`
- Dáta sú posielané ako dvojice identifikátor = hodnota, čo je veľmi nešikovné v prípade odosielania textu, často zdĺhavé názvy premenných uberajú na kapacite textu, ktorý sa chystáme odoslať.
- Query string je slovník, prostredníctvom ktorého síce dokážeme odoslať, dáta no už nedokážeme uchovať žiadne vzťahy medzi nimi, čo kladie zvýšené nároky na tvorbu klientskej a serverovej časti aplikácie, pričom každá z častí musí poznať a implementovať všetky vzťahy medzi jednotlivými informáciami.

Problém s obmedzenou kapacitou, ktorú nám query string ponúka môžeme riešiť použitím metódy POST namiesto metódy GET a dáta prenášať v tele tejto http požiadavky, čo nám štandardne ponúka 2MB priestoru, ale táto veľkosť sa dá zmenou nastavenia serveru modifikovať až na 2GB(server Apache2).
Problém neštruktúrovateľnosti dát sa dá riešiť prostredníctvom odosielania dát vo formáte XML.
Jazyk AS 3.0 obsahuje balíček flash.xml ktorý ponúka možnosti spracovania xml dokumentov tak ako sme na to zvyknutý z iných OOP jazykov ako sú C# a Java.
Z toho dôvodu sa nám ponúka možnosť integrácie AS 3.0 a serverových technológií prostredníctvom dát vo formáte XML. Na túto integráciu môže byť využitý dočasný XML súbor umiestnený na serveri, ale v takomto prípade je potreba riešiť problém súčasného prístupu k nemu , alebo môžu byť XML dáta integrované do tiel POST metód a prenášané po sieti.

3.3. Integrácia AS 3.0 a serverových technológií prostredníctvom dát vo formáte XML [6]

Rovnako ako pri použití metódy `GET` na prenos dát prostredníctvom query stringu, tak aj pri využití metódy `POST` na prenos XML dát je postup na strane AS 3.0 rovnaký bez závislosti na použitej serverovej technológií. Tento fakt poukazuje na to, aký komplexný a jednoduchý je jazyk AS 3.0.
Principiálne sa využitie metódy `POST` len veľmi málo líši oproti použitiu metódy `GET`, ktorá bola popisovaná v predchádzajúcej časti.

Na rozdiel od predchádzajúce kódu nie je nutné použitie triedy `URLVariables` a inštancií triedy `URLRequest` je potreba nastaviť ešte jeden paramter a to konkrétne `URLRequest.contentType` na hodnotu `"text/xml"`, tento príznak určuje aký typ dát bude nesený telom `http Post` požiadavky a pochopiteľne typ metódy určený parametrom `URLRequest.method` zmeniť na `URLRequestMethod.POST`.
Ďalším rozdielom je pripojenie dát k našej požiadavke. Toto realizuje prostredníctvom `URLRequest.data = mojXMLDoc`.
Pri používaní metódy post prenášajúcej v tele XML dáta môžu nastať 4 základne výnimky, ktoré treba ošetriť a to:

---

[5] Viac o query stringu sa nachádza na: **[10].**
[6] Táto časť textu je inšpirovaná **[11]** , viac k danej problematike dostupné na **[12]** a **[13]**.

`TypeError` - snažíme sa odoslať/prijať iný typ ako bol nastavený v argumente `URLRequest.contentType`,
`ArgumentError` - neplatné URL,
`SecurityError` - nastavenie bezpečnosti nepodporuje metódu post ,
`IOErrorEvent` - táto výnimka je riešená vlastnou udalosťou počas príjmu / odosielania došlo k chybe.
Zvyšný postup a funkcionalita je zhodná s prípadom užitia metódy `GET`, ktorá bola popísaná v predchádzajúcej kapitole 3.1.0.

Nasledujúci AS 3.0 kód odošle na server XML dáta, server ich prečíta a odošle späť, následné je dané XML zachytené a spracované :

3.3.0. AS 3.0 : Metóda Post a odoslanie dát prostredníctvom XML [7]

```
function postXmlData()
{
      var dataXML:XML =
                  <logins>
                        <person>
                              <username>SomeName</username>
                              <password>SomePass</password>
                        </person>
                  </logins>;
      var request:URLRequest = new URLRequest("http://MyUrl.aspx ");
      request.contentType = "text/xml";
      request.data = dataXML;
      request.method = URLRequestMethod.POST;
      var loader:URLLoader = new URLLoader();
      try {
            loader.addEventListener(Event.COMPLETE, onComplete,
            false, 0, true);
            loader.addEventListener(IOErrorEvent.IO_ERROR, onIOError,
            false, 0, true);
            loader.load(request);
      }
      catch (error:ArgumentError){
            trace("An ArgumentError has occurred.");
      }
      catch (error:SecurityError){
            trace("A SecurityError has occurred.");
      }
      var xmlResponse:XML;
      function onComplete(evt:Event):void {
            try {
                  xmlResponse = new XML(evt.target.data); [8]
                  trace(xmlResponse.person[0].username);
                  removeEventListener(Event.COMPLETE, onComplete);
                  removeEventListener(IOErrorEvent.IO_ERROR,
                  onIOError);
            }
```

[7] Popis funkcionality kódu sa nachádza v časti 3.3. a 3.1.0.
[8] Podrobné informácie o spôsobe práce s XML prostredníctvom AS 3.0 na: **[14]**

```
            catch (err:TypeError) {
                  trace("error:\n" + err.message);
            }
      }

      function onIOError(evt:IOErrorEvent):void {
            trace("error\n" + evt.text;
      }
}
```

3.3.1. Spracovanie dát na servery a následná odpoveď späť.

Príklady zachytenia odoslaných XML dát na server a ich následné odoslanie späť:

3.3.1.0. ASP.Net :

```
%@ Page Title="test page" Language="C#"%>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.Web" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<script runat="server">
    public void Page_Load(object sender, EventArgs e)
    {
        if(Request.ContentType == "text/xml")
        {
           //vytvorenie XMLDomObjektu
            XmlDocument doc = new XmlDocument();
           //nacitanie XML dát z Postu
            doc.Load(Request.InputStream);
           //odpoved odoslanie obsahu XMLDomObjektu
            Response.Write(doc.OuterXml);
            Response.End();// koniec odpovedi
        }
</script>
```

Metóda `Page_Load` je obslužná metóda udalosti načítania danej stránky, bude volaná vždy, keď AS 3.0 odošle požiadavok na dané `URL`. Úvodna podmienka zisťuje či v tele požiadavky – Request sa nachádzajú dáta vo formáte XML. Ak áno, vytvorí sa `XMLDomObject` do ktorého sa dáta načítajú pomocou metódy `Load`, tá berie ako parameter `Request.InputStream` čo je datový stream požiadavky, resp. telo Post metódy. Napokon do odpovede `Response` vpíšeme našu odpoveď prostredníctvom metódy `Write`, ako parameter tejto metóde dáme `XMLDomObject.OuterXml`, čo sú interné dáta daného `xml` bez špecifikácie samotného `XMLDomObjectu`. V podstate sa jedná o xmltagy a dáta medzi nimi.
Kedže sme pracovali so streammi treba odpoveď aj ukončiť cez volanie `Response.End()`.

3.3.1.1. PhP : [9]

```
<?php
if (isset($GLOBALS["HTTP_RAW_POST_DATA"]))//ak nie je null
{
//    $xml = xmldoc($GLOBALS["HTTP_RAW_POST_DATA"]);
//   vytvorenie XMLDom dokument kód je funkčný aj bez toho
    $xml = $GLOBALS["HTTP_RAW_POST_DATA"]; //odoslane XMl data

    echo($GLOBALS["HTTP_RAW_POST_DATA"]); //odoslanie dát späť
//  echo($xml); // druha možnosť odpovede ak sme použili DOM
}
?>
```

PhP uchováva zachytené dáta metódou `Post` v `$GLOBALS["HTTP_RAW_POST_DATA"]`, tieto dáta priradíme nejakej premennej a následne odošleme späť príkazom `echo`. Pokiaľ by bolo potreba dané xml na strane servera spracovávať, treba ho priradiť `XMLDomObjectu` v PhP `xmldoc`.

3.3.1.2. JSP [10] :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class myInfo extends HttpServlet{
          private static final String CONTENT_TYPE = "text/xml";
          public void init(ServletConfig config) throws
          ServletException
          {
                    super.init(config);

          }
          public void doGet(HttpServletRequest request,
          HttpServletResponse response) throw
          ServletException, IOException
          {
               response.setContentType(CONTENT_TYPE);
               PrintWriter out = response.getWriter();
          }
          public void doPost(HttpServletRequest request,
          HttpServletResponse response) throws
          ServletException, IOException{
               response.setContentType(CONTENT_TYPE);
               PrintWriter out = response.getWriter();
               out.write(request.getInputStream());
               out.close();
          }
}
```

---

[9] Podrobnejšie informácie na: **[15]**

[10] Viac o JSP serverovej kostre na **[7]** a **[8]** a v časti 3.1.1.1.

Toto je štandardná kostra jednoduchého JSP serveru, AS 3.0 kód využívajúci tento server neimplementuje metódu `GET`, preto ju nie je potreba riešiť. Upravená je len metóda `doPost`, kt. spracováva Post požiadavky. Najprv je potreba odpovedi nastaviť formát prostredníctvom volania `response.setContentType(format)`, potom vytvoríme inštanciu writeru pre odpoveď `PrintWriter`, priradíme mu štandardný writer odpovedi `response.getWriter()`, potom zapíšeme prichádzajúce dáta, kt. získame z `request.getInputStream()` do vytvoreného `Writera`, napokon zatvoríme stream. `Writer.close()`.

3.4. Socketové spojenie AS 3.0 – Java [11] :

Námetom pre vytvorenie tejto kapitoly bol článok **[16]**.
Kedže je AS 3.0 silne inšpirovaný jazykom Java, rozhodli sa jeho tvorcovia doň pridať triedu `XMLSocket` pre uľahčenie spojenia medzi AS 3.0 a Javou prostredníctvom tzv. socketovej komunikácie.
Komunikačné prostriedky na strane Javy sú umiestnené v balíčku : `org.xsocket.connection`. Tento je voľne stiahnuteľný a šíriteľný v podobe `.jar` súboru napr. na adrese:
`http://grepcode.com/snapshot/repo1.maven.org/maven2/org.xsocket/xSocket/2.2`
Pre funkčnosť nasledujúceho kódu som využil verzie 2.2, ale je možné použiť aj novšie.
Princíp socketovej komunikácie je zjednodušene takýto:
Spustí sa Java server bežiaci na určitej adrese načúvajúci na určenom porte. Klientska časť otvorí spojenie na daný server. Skrz toto spojenie je možné posielať dáta, spojenie ostáva otvorené, kým nie je ukončené zo strany serveru alebo klienta.

3.4.0. Kód Java serverovej aplikácie:

Java server – Main :

```
package socketcom;
import org.xsocket.connection.*;
public class Main{
    protected static IServer srv = null;
    public static void main(String[] args){
        try{
            srv = new Server("127.0.0.1", 8090, new
                xSocketDataHandler());
            srv.run();
        }
        catch(Exception ex){
            System.out.println(ex.getMessage());
        }
    }
    protected static void shutdownServer(){
        try{
            srv.close();
        }
        catch(Exception ex){
            System.out.println(ex.getMessage());
        }
    }}
```

[11] Podrobná technická špecifikácia, dokumentácia a príklady na: **[17]**

Pre využívanie socketov je potreba importovať balíček import `org.xsocket.connection.*`
Na začiatku je potreba vytvoriť si inštanciu Internetového serveru – `Iserver`. Potom treba tento server inicializovať a to konkréne volaním konštruktoru serveru `new Server`, ktorému parametrom odovzdáme `IP, port` a `DataHandler` t.j. kód zabezpečujúci spracovanie dát. Musí to byť trieda implementujúca rozhranie `IDataHandler`. Napokon treba server zapnúť príkazom `IServer.run`(). Ďalej treba implementovať metódu na vypnutie serveru, vypnutie serveru prebehne volaním `IServer.close()`.

Trieda zabezpečujúca spracovanie dát a vypnutie serveru zo strany klienta:

```
package socketcom;
import java.io.IOException;
import java.nio.BufferUnderflowException;
import java.nio.channels.ClosedChannelException;
import org.xsocket.*;
import org.xsocket.connection.*;
public class xSocketDataHandler implements IDataHandler
{
     public boolean onData(INonBlockingConnection nbc) throws
     IOException, BufferUnderflowException,
     ClosedChannelException, MaxReadSizeExceededException
     {
        try
        {
            String data = nbc.readStringByDelimiter("\0");
            nbc.write(data + "\0");

            if(data.equalsIgnoreCase("SHUTDOWN"))
                Main.shutdownServer();
        }
        catch(Exception ex)
        {
            System.out.println(ex.getMessage());
        }
        return true;
     }
}
```

Toto je obslužná trieda, kt. spracováva prichádzajúce dáta, aby mohla správne fungovať musí implementovať rozhranie `IDataHandler`. Dáta prúdiace medzi socketmi sa nachádzajú v inštancii triedy `INonBlockingConnection`, ktorá ako každý iný stream ma metódy `read` a `write`, konkrétne sa používa modifikácia metódy `read` a to `readStringByDelimiter`.
Poznámka - AS 3.0 využíva znak "\0" ako oddeľovač resp. posledný znak správy. Tak je možno presne určiť koniec správy.
Prečítané dáta odošleme späť skrz metódu `write`, znova aby AS 3.0 rozpoznalo koniec správy je potreba pridať oddeľovač `"\0"`.
Ak sú prichádzajúce dáta resp. správa `"SHUTDOWN"`, je volaná metóda ukončujúca činnosť serveru.

3.4.1. AS 3.0 Klientska aplikácia :

```
function socketCon(param:String)
{
      var xmlSocket:XMLSocket = new XMLSocket();
      xmlSocket.connect("127.0.0.1", 8090);

      xmlSocket.addEventListener(DataEvent.DATA,
      onIncomingData);

      function onIncomingData(event:DataEvent):void
      {
            trace("[" + event.type + "] " + event.data);
      }
      xmlSocket.send(param);
      //odoslatie slova shutdown vypne server
}
```

Najprv je potreba vytvoriť inštanciu triedy `XMLSocket`. Potom realizovať spojenie volaním metódy `XMLSocket.connect`, ktorá príjme ako parameter adresu a port. Ďalej som pridal `DataEvent`, kt. je volaný v prípade, že socketom prichádzajú dáta, a obslužnú metódu, ktorá tieto dáta zachytáva. Odosielanie dát skrz socket sa uskutočňuje prostredníctvom metódy `send("data k odoslaniu")` triedy `XMLSocket`.

3.5. Zhrnutie – Spojenie AS 3.0 a serverových technológií schopnej konektivity s DB.

Princíp komunikácie zo strany AS 3.0 je vždy rovnaký a nezáleží na použitej serverovej technológií. Z toho dôvodu nie je možné určiť najlepšiu metódu na prepojenie AS 3.0 a báze dát. Celý problém sa presúva na porovnávanie jednotlivých technológií ako sú PhP, ASP.Net a Java. To je problém, ktorý nie je možné rozhodnúť, pretože každá z uvedených technológií má svoje klady a zápory a taktiež záleží na použitej databázovej technológií.
Jednotlivé prípady je potreba posudzovať individuálne, zvážiť všetky pre a proti daného riešenia a v neposlednej rade schopnosti a skúsenosti vývojára, ktorý sa chystá problém riešiť.
Všeobecne platí pokiaľ realizujeme projekt, pri ktorom dochádza ku kontinuálnemu prenosu dát, napríklad pri hre kde sú prenášané dáta do db. a z db. počas dlhšej doby, t. j. čas, počas ktoré je hra zapnutá, je vhodné využiť socketovú komunikáciu a každému užívateľovi takejto hry prideliť serverový port, ktorý jeho požiadavky obslúži.
Pre informačné systémi, pri ktorých dochádza k ojedinelým vstupom do db. na základe požiadaviek klienta je lepšie využiť prenosu dát prostredníctvom Http metód GET a POST. Medzi jednotlivými metódami treba voliť na základe veľkosti, štruktúri a verejnosti odosielaných dát.
Obslužnú technológiu treba voliť na základe použitého typu db., technických prostriedkov a skúseností vývojára, kt. realizuje daný projekt.
Osobne za najpoužívanejšie a najefektívnejšie databázy považujem Microsoft SQL a Oracle, s kt. je veľmi jednoduché zabezpečiť konektivitu prostredníctvom jazykov Visual Basic a C#.

K prepojeniu databáz Oracle a programovacích jazykov rodiny .Net sa používa Oracle Data Acess Component(ODAC) , kt. umožňuje jednoduché a efektívne spracovanie dát z databáze. Práve preto si pre implementáciu svojho projektu vyberám Databázu Oracle 11g a ako prostredníka jazyk C#. Na prepojenie týchto dvoch využijem spomínaný ODAC.

V nasledujúcej kapitole je zdokumentovaný demonštračný informačný systém, pri ktorého tvorbe som využil väčšinu v predchádzajúcich kapitolách opísaných princípov a programovacích metód.

Tento informačný systém slúži k praktickému overeniu nadobudnutých znalostí a otestovaniu teoretických poznatkov v praxi pri spracovaní väčších a štruktúrovaných dát.

4.0. Informačný systém – Knižnica

4.1. Ciele vytvorenia Informačného systému knižnica:

Hlavným cieľom je vytvoriť komplexný Informačný systém(ďalej už len IS) pokrývajúci všetky potreby zamestnancov a klientov knižnice, úplné zjednodušenie práce zamestnancov knižnice, skvalitnenie klientskych služieb poskytovaných knižnicou a presun funkcionality bežnej knižnice do priestoru internetu umožňujúci klientovi využívanie bežných služieb knižnice z pohodlia svojho domova.
Vedľajším cieľom je postupom času hromadiť údaje o jednotlivých pôžičkách a tieto údaje štatisticky spracovávať a získať tak informácie o obľúbenosti určitých autorov, žánrov, diel, zistiť záujmy rôznych vekových skupín a pod.
Získavať a zaznamenávať sa budú aj užívateľské komentáre a hodnotenia jednotlivých diel. Z nahromadených dát budú vytvárané rebríčky(obľúbenosť, hodnotenie) a tie budú slúžiť ako doplnkové informácie pre užívateľov.

4.2. Predpokladaný obsah dát uchovaných v IS knižnica:

Tento systém bude obsahovať zoznam autorov, ktorý napísali aspoň jedno dielo, ktoré knižnica vlastní a rozširujúce informácie o jednotlivých autoroch.
Tieto informácie budú slúžiť ako doplnkové informácie pre užívateľa a umožnia katalogizáciu jednotlivých diel podľa ich autorov.

V systéme budú uchovávané informácie o jednotlivých nakladateľstvách, ktoré publikovali aspoň jedno dielo vlastniace danou knižnicou.
Budú to informácie ako napr. kontaktné údaje daného nakladateľstva, jeho meno a pod. Tieto informácie budú potrebné napríklad pre zisk autorských práv pre „online“ publikovanie niektorých diel, doobjednanie ďalších kusov daného diela, reklamácie a pod.

Systém bude uchovávať zoznam všetkých známych žánrov kníh s krátkym popisom.
Tieto dáta budú doplnkovou informáciou pre užívateľa a umožnia ďalšiu katalogizáciu diel podľa svojho žánru.

Ďalej bude systém uchovávať informácie o samotných dielach, ktoré daná knižnica vlastní ako napríklad názov diela, rok vydania, aktuálny stav(Avaliability [0 – nedostupná a 1 – dostupná]), popis a pod.
Každé dielo bude mať prostredníctvom identifikátoru(referencie) pripojené informácie o svojom autorovi, žánri, nakladateľstve a pod.

Ďalšími informáciami uchovanými v tomto systéme budú informácie o klientoch knižnice t.j. ich id. v rámci knižničného IS., „nickname“ : meno, pod ktorým budú vystupovať v rámci IS , meno a priezvisko, rok narodenia, kontaktné údaje.

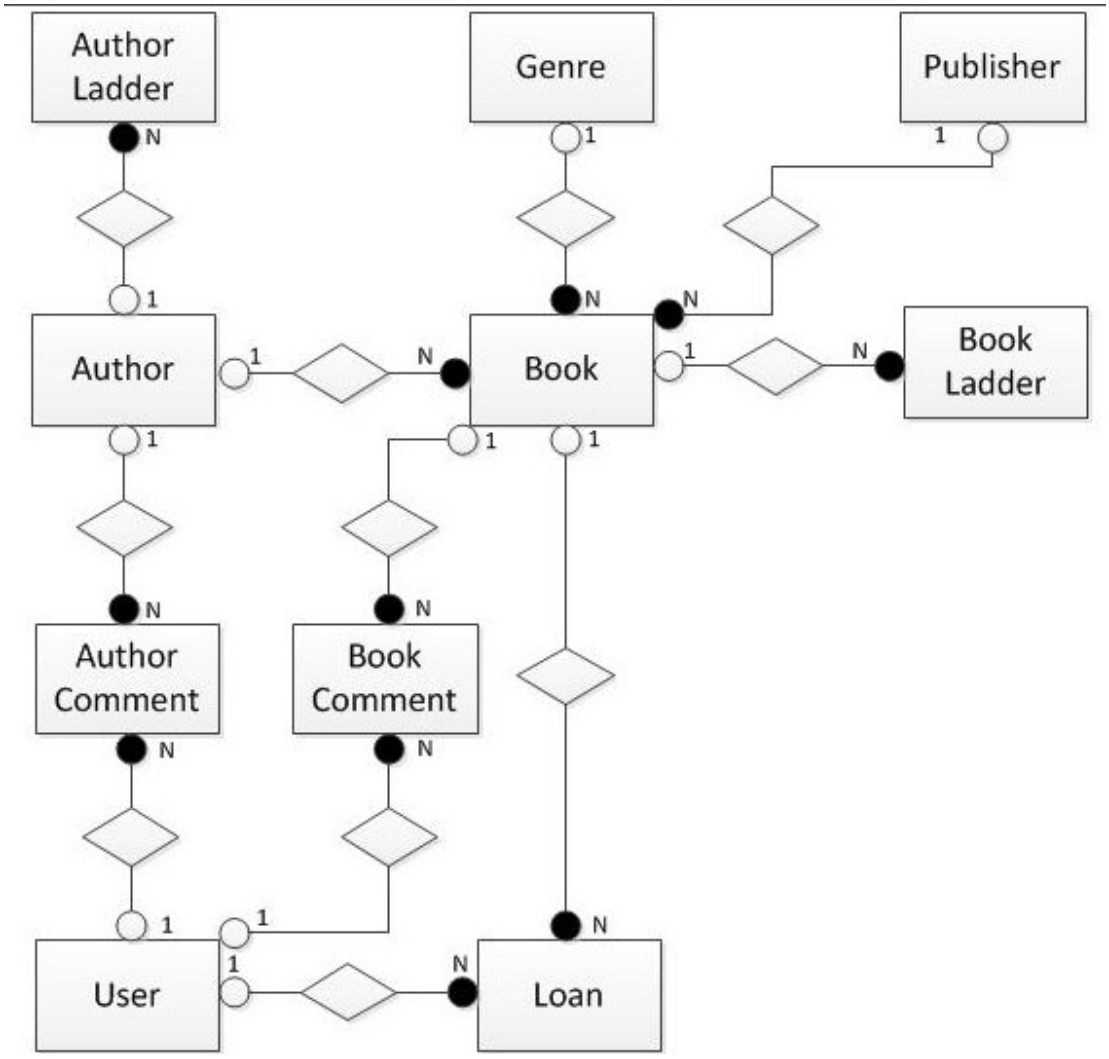
Užívatelia budú mať možnosť vkladania komentáru k danému dielu a autorovi súčasne s komentárom budú vkladať aj svoje hodnotenie(číselná hodnota od 0 do 10 vyjadrujúca ich spokojnosť).

Tieto komentáre a hodnotenia budú uchovávane v IS a budú prezentované ďalším užívateľom ako doplnkové informácie.

Po každom roku budú z rôznorodých informácií uchovanými v IS. týkajúcich sa autorov a jednotlivých diel vytvorené rebríčky pre daný rok. Budú zostavované na základe užívateľských hodnotení a počtu vypožičaní resp. prezretí(v prípade „online“ prezentácie diela). Obsahovať bude informácie o roku pre ktorý je platný, diele alebo autorovi na ktoré/ho sa vzťahuje, jeho dve poradové miesta(na základe hodnotenia a na základe počtu pôžičiek resp. prezretí) na ktorých sa nachádza, priemerné dosiahnuté hodnotenie za daný rok a počet požičaní/prezretí za daný rok.

Poslednými dátami uchovávanými v tomto systéme budú podrobné záznami o všetkých vykonaných pôžičkách, t.j. id klienta, ktorý si knihu vypožičal, id knihy, ktorá bola zapožičaná respektíve prehliadnutá(v prípade „online“ prezentácie diela), dátum vypožičania a vrátenia knihy.

4.3. ER diagram – dátová schéma:



4.4. Prístup a využitie dát uloženým v IS knižnica:

- **Administrátorský prístup -** *plný prístup* **:** možnosť pridávať, mazať a úplne editovať všetky záznami (dáta) uchované týmto IS prostredníctvom administrátorského rozhrania, ktoré bude bežným užívateľom neprístupné.
  Tento prístup bude mať len hlavný administrátor a osoby ním poverené vykonávať administračné práce na celom IS(napr. zamestnanci).
- **Klientsky prístup –** *náhľadovo - informačný prístup (tzv. Spectator)* **:**
  - možnosť náhľadu na informácie týkajúce sa klientskeho účtu :
    - ❖ Informácie o jednotlivých pôžičkách spojených s daným klientskym účtom.
    - ❖ Náhľad do zoznamu diel a autorov spojený s informáciami o dostupnosti daného diela. Možnosť zobrazenia profilov jednotlivých diel a autorov, kde sa budú nachádzať rôzne užívateľské komentáre, hodnotenia a doplňujúce informácie o jednotlivých dielach.
  - možnosť náhľadu a editácie užívateľských dát zadávaných pri registrácií. Každá editácia bude spojená s validáciou zadávaných dát(zamedzenie krádežiam užívateľských účtov).
  - možnosť pridávania užívateľských komentárov a hodnotení.
  - možnosť „online“ prehliadky niektorých diel v čítačke, prípadne možnosť náhľadu iba na časť diela(niečo ako ochutnávka, užívateľ sa potom rozhodne či si dielo požičia, bude záležat na autorských právach či daná knižnica bude mať práva na „online“ publikovanie) .

  Administrátorské rozhranie bude pre tento prístup nedostupné, preto akákoľvek možnosť editácie mazania alebo pridávania dát bude zamedzená(okrem užívateľských údajov zadaných pri registrácií, komentárov a hodnotení).
  Tento prístup budú mať všetci klienti knižnice.

4.5. Účastníci IS knižnica:

- **Hlavný administrátor** – Osoba plne znalá celého IS knižnica, jeho úlohou je dohliadať na celkový IS a po jeho uvedení do prevádzky zabezpečuje plynulý chod IS a jeho údržbu, prípadne modifikácie. Jeho úlohou je riešiť a upravovať formát uložených dát, nezaoberá sa samotnými uloženými dátami v IS.
- **Administrátor - Zamestnanec** - Osoba ovládajúca formát a štruktúru dát uložených v IS, jeho hlavnou úlohou je spracovávať dáta a správnym spôsobom ich ukladať v IS. Uložené dáta musí byť tiež plne schopný interpretovať, prípadne editovať.
  Pracuje so všetkými dátami uloženými v IS vrátane osobných klientskych dát(preto nutné pri registrácií žiadať klienta o súhlas so spracovaním osobných údajov v zmysle § 16 zákona č. 428/2002 Z. z o ochrane osobných údajov.
- **Klient** – Osoba po registrácií schopná editovať a pozorovať svoje osobné a verejné dáta(všetko okrem osobných údajov iných klientov) a pridávať komentáre a hodnotenia.

4.6. Funkčná špecifikácia IS knižnica:

4.6.0. Súbor funkcií:

- **Prihlásenie:**
  Účastníci : Administrátor/ Zamestnanec, Klient
  Popis: Funkcia vykoná prihlásenie do systému.
  Vyžaduje sa: Aktívny IS, vyplnenie osobných údajov(nick,heslo) zhodných s uloženými pri registrácií.
  Výsledok: Užívateľ je prihlásený do IS a sprístupnia sa mu nové funkcie na základe typu prihláseného účtu.
  Rozšírenia:
  Registrácia – pred prihlásením je potreba sa najprv registrovať užívateľ vyplní osobné údaje, prebehne validácia a pokiaľ sú dáta validné a užívateľské meno voľné, vytvorí sa záznam o užívateľovi, ktorý sa zapíše do tabuľky „user“.
  Zamestnanecký účet musí vytvoriť hlavný administrátor.

- **Správa účtu**
  Účastníci : Klient
  Popis : funkcia IS umožňujúca zobrazenie užívateľských údajov, ich následnú editáciu a obnovenia hesla.
  Vyžaduje sa : Aktívny IS, byť prihlásený ako klient.
  Výsledok: Užívateľ je schopný pracovať so svojimi osobnými údajmi bez nutnosti zásahu administrátora/zamestnanca.
  Rozšírenia :
  Editácia osobných údajov – užívateľ môže modifikovať svoje osobné údaje zadané pri registrácií, za predpokladu, že vyplnené nové údaje(„nickname“, meno, priezvisko, rodné číslo, email, heslo, číslo Občianskeho preukazu, kontaktná adresa) spĺňajú kritéria validácie.
  Obnova hesla – užívateľ si môže nechať zaslať svoje heslo na email vyplnený pri registrácií, prípadne môže zažiadať o vygenerovanie nového hesla, ktoré sa mu následne zašle na email. Aby bolo toto možné vyžaduje sa správne vyplnený „nickname“ alebo email.

- **Vyhľadávanie:**
  Účastníci: Zamestnanec(všetky dáta), Klient(len verejné dáta t.j. dáta mimo osobné údaje iných užívateľov)
  Popis: Funkcia umožňujúca vyhľadávanie dát v databáze a následne ich zobrazenie.
  Vyžaduje sa: Aktívny IS, vyplnenie výrazu na základe ktorého prebehne vyhľadávanie(priezvisko, autora, titul, ... a miesto kde sa bude vyhľadávať).
  Výsledok: Zoznam objektov zhodujúcich sa s hľadaným výrazom a jeho/ich(v prípade nájdenia celej kolekcie takýchto objektov) následne zobrazenie.

- **Zobrazenie profilu diela alebo autora:**
  Účastníci: Klient
  Popis: Funkcia umožňujúca zobrazenie objektu(autor alebo dielo) so všetkými dátami k nemu viazanými(komentármi a hodnoteniami) na jednom mieste .
  Vyžaduje sa: Aktívny IS, vybranie daného objektu zo zoznamu(kliknutím), byť prihlásený ako klient.
  Výsledok: Zobrazenie informácií a dát vzťahujúcich sa k danému objektu na špeciálnom mieste s jednoduchým „interface“ slúžiacim k vyplneniu vlastných komentárov a hodnotení.
  Rozšírenia :
  Pridanie komentára a hodnotenie – Pripojí k danému objektu užívateľský komentár. K tomuto účelu sa na profilovej stránke bude nachádzať jednoduchý „textbox“ a tlačítko odoslať, ktoré pripojí k danému objektu užívateľský text spolu s „nickname“ užívateľa. Pri kliknutí na „odoslať“ sa vytvorí záznam v tabuľke „author comment / book comment“ v závislosti na objekte pri ktorom sa komentár pridáva. Ďalej sa pripojí k danému objektu užívateľské hodnotenie. K tomuto účelu sa na profilovej stránke bude nachádzať jednoduchý „drop down list“ a tlačítko odoslať, ktoré pripojí k danému objektu užívateľské hodnotenie(vybranú hodnotu z drop down listu). Pri kliknutí na „odoslať“ sa modifikuje záznam v tabuľke „author comment / book comment“ v závislosti na objekte pri ktorom sa komentár pridáva a to konkrétne hodnota „ranking“.

- **„Online“ prehliadka diela:**
  Účastníci: Klient
  Popis: V IS bude implementovaná „online“ čítačka, ktorá bude umožňovať prezeranie diel priamo na stránkach. Pre možnosť „online“ prezentácie diel bude knižnica povinná získať práva na zverejnenie obsahu diela v zmysle zákona č. 618/2003 z.z. a v súlade s Bernskou dohodou o ochrane literárnych a umeleckých diel. Pri dielach, na ktoré knižnica nezíska práva na zverejnenie bude v čítačke len veľmi obmedzený obsah, ktorý bude slúžiť ako upútavka pre užívateľa.
  Pri „online“ prehliadke dôjde k automatickému vytvorenia záznamu v tabuľke „Loan“, ktorý bude slúžiť k štatistickému spracovaniu.
  Vyžaduje sa: Aktívny IS, prihlásenie ako užívateľ, prehliadač podporujúci adobe flash.
  Výsledok: Užívateľ je schopný prezerať si diela z pohodlia vlastného domova stačí mu internetové pripojenie a platný užívateľský účet(account).

- **Rezervácia diela:**
  Účastníci: Klient
  Popis: V prípade že má klient záujem určité dielo si fyzicky požičať a dielo nie je momentálne v knižnici, môže zažiadať o jeho rezerváciu. Po vrátení diela iným užívateľom prípadne po zakúpení ho knižnicou si môže dielo vyzdvihnúť alebo si ho

nechať poslať. Záleží na dohode so zamestnancom knižnice, ktorý bude rezerváciu spracovávať .
Vyžaduje sa: Aktívny IS, prihlásenie ako užívateľ.
Výsledok: Užívateľ má možnosť rezervácie akéhokoľvek diela a jeho následné vypožičanie, prípadne si môže dohodnúť jeho zaslanie kuriérom.

- **Vytvorenie pôžičky**
  Účastníci: Zamestnanec
  Rozšírenia:
  Vrátenie knihy– pri vrátení knihy sa automaticky do príslušného záznamu doplní dátum vrátenia knihy, zamestnanec musí len identifikovať pôžičku a kliknúť na tlačítko „return book“.
  Popis: Zamestnanec v prípade fyzického vypožičania vytvorí manuálne záznam o pôžičke s prázdnou resp. s nulovou hodnotou u parametru „DateOfReturn“, ktorú modifikuje pri vrátení knihy .
  Vyžaduje sa: Aktívny IS, prihlásenie ako administrátor/zamestnanec.
  Výsledok: Vytvorenie záznamu o fyzickej pôžičke knihy a jej vrátenie.

- **Pridávanie:**
  Účastníci: Zamestnanec
  Jednotlivé pridávanie dát do rôznych častí IS(vytváranie nových záznamov v jednotlivých tabuľkách databáze).
  Popis: Funkcia umožňuje pridanie dát do databáz, ako napríklad vytvorenie nového žánru, nakladateľstva, autora, diela.
  Vyžaduje sa: Aktívny IS, vyplnenie všetkých potrebných údajov pre úspešné pridanie dát do IS, záleží na konkrétnom type dát ktoré bude pridávané. Bude sa jednať o dáta ako napríklad(meno ,priezvisko ,dátum narodenia, žáner , nakladateľstvo, titul, užívateľ, ...) v správnom formáte.
  K tomuto účelu bude vytvorený jednoduchý špeciálny „interface“ dostupný len ľudom s administrátorsko-zamestnaneckými právami.
  Výsledok: Modifikovaná množina dát IS.

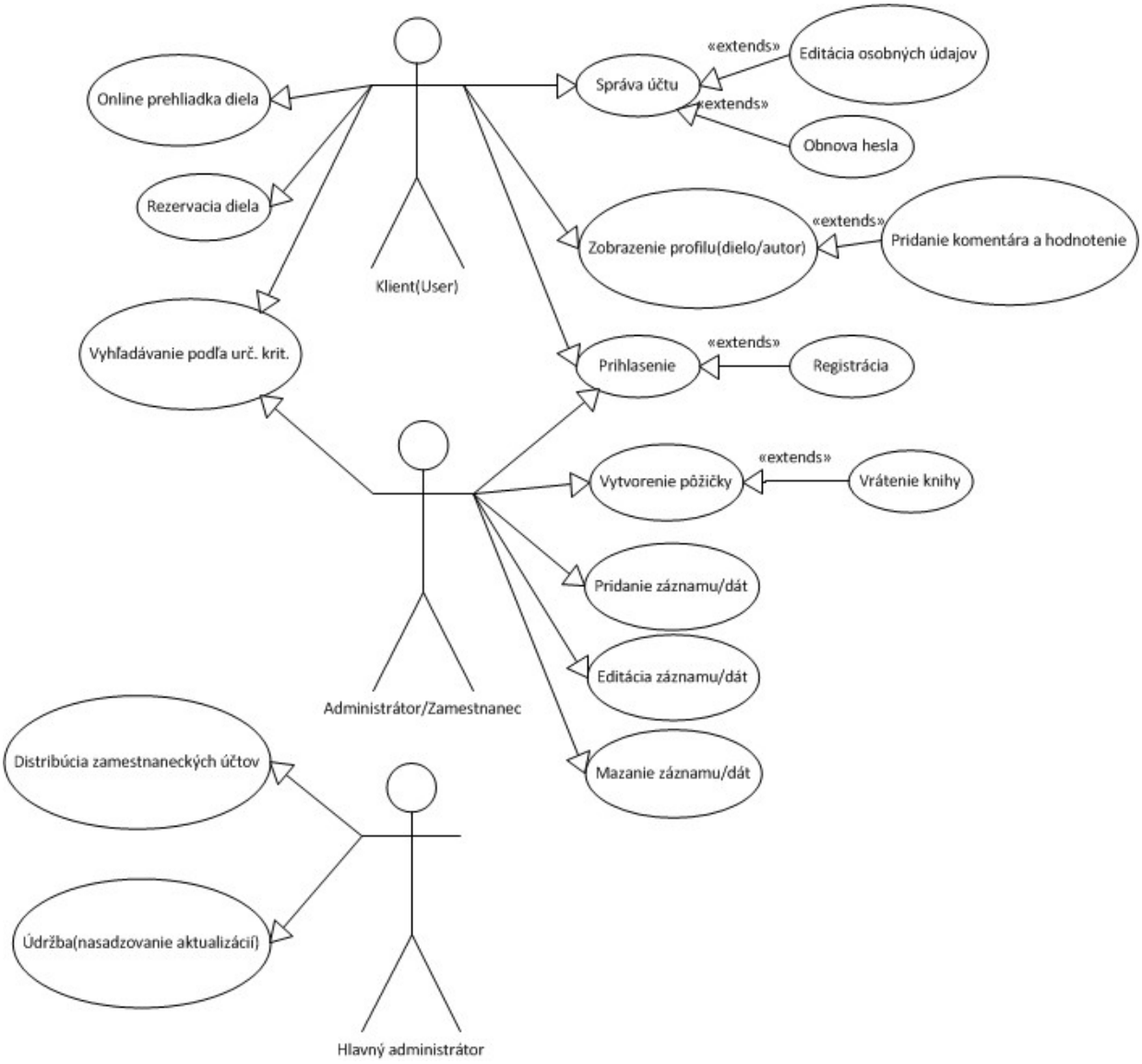
- **Editácia:**
  Účastníci: Zamestnanec
  Editácia rôznych dát v rôznych častiach IS.
  Popis: Funkcia umožňuje editáciu(„update“) jednotlivých dát v databáze
  Vyžaduje sa: Aktívny IS, dohľadanie záznamu, vyplnenie všetkých potrebných údajov ako napríklad (meno, priezvisko, dátum narodenia, žáner, nakladateľ, titul, užívateľ, ...) potrebných pre úspešné pridanie dát do IS v správnom formáte.
  K tomuto účelu bude vytvorený jednoduchý špeciálny „interface“ dostupný len ľudom s administrátorsko-zamestnaneckými právami.
  Výsledok: Modifikovaná množina dát IS.

- **Mazanie:**
  Účastníci: Zamestnanec
  Popis: Manuálne zmazanie dát prípadne dátových štruktúr v prípade chyby IS alebo zamestnanca.
  Vyžaduje sa: Aktívny IS, dohľadanie záznamu. K tomuto účelu bude vytvorený jednoduchý špeciálny „interface“ dostupný len ľudom s administrátorsko-zamestnaneckými právami.
  Výsledok: Modifikovaná množina dát v IS.

- **Priraďovanie práv užívateľom:**
  Účastníci: Hlavný administrátor
  Popis: Distribúcia zamestnaneckých prípadne administrátorských práv na jednotlivé účty.
  Výsledok: Modifikované práva užívateľského účtu prípadne nový účet s pridelenými právami

- **Údržba:**
  Účastníci: Hlavný administrátor
  Pridávanie nových dátových štruktúr alebo funkcií na základe požiadaviek zákazníka.
  Update formátu dát v IS.
  Zálohovanie, prečisťovanie a optimalizácia.
  Popis: Všetky činnosti smerujúce k skvalitneniu služieb a funkcionality IS
  Výsledok: Optimálny chod IS.

4.6.1. UseCase Diagram IS knižnica:



4.7. Technická špecifikácia IS knižnica:

4.7.0. Predpokladaný objem dát a odhadované počty záznamov:

User (Nick[Varchar(20)],FirstName[Varchar(20)],LastName[Varchar(30)],userId[Smallint], Email[(Varchar30)], Password[Varchar(20)], BirthDate[Datetime], Country[Varchar(30)], City[Varchar(30)], PostalCode[Varchar(5)], Street[Varchar(30)], Numb[Int])
Sum = 2 + 20 + 20 + 30 + 30 + 20 + 8 + 30 + 30 + 5 + 30 + 4 = 229 B
Predpokladá sa 500 – 600 užívateľov t.j. 229 x 600 = 137 400 B t.j. cca. 0,13 MB

Autor(FirstName[Varchar(20)],LastName[Varchar(30)],authorId[Smallint], BirthDate[Datetime], Descript[Varchar(255)])
Sum = 20 + 30 + 2 + 8 + 255 = 315 B
Predpokladá sa cca 10 000 záznamov t.j. 315 x 10 000 = 3 150 000 B t.j. cca. 3,00 MB

Publisher(CompanyName[Varchar(40)], companyId[Smallint], Country[Varchar(30)], City/Town[Varchar(30)], PostalCode[Varchar(5)], Street[Varchar(30)], Numb[Int], Email[Varchar(30)], PhoneNumber[Varchar(13)])
Sum = 40 + 2 + 30 + 30 + 5 + 30 + 4 + 30 + 13 = 184 B
Predpokladá sa 40 až 100 nakladateľstiev t.j. 184 x 100 = 18 400 B t.j. cca. 0,02 MB

Genre(Genre[Varchar(30)],genreId[Smallint], Descri[Varchar(255)])
Sum = 30 + 2 + 255 = 287 B
Predpokladá sa cca 40 rôznych žánrov t.j. 32 x 40 = 11 480 B t.j. cca 0,01 MB

Book(TitleName[Varchar(50)],bookId[Int],YearOfPublishment[Date],Autor_Id[Smallint], Genre_Id[Smallint],Publisher_Id[Smallint], Avaliable[Bit], Desc[Varchar(255)], bookUrl[Varchar(50)])
Sum = 50 + 4 + 4 + 2 + 2 + 2 + 1 + 255 + 50 = 320 B
Predpokladá sa až 12 000 záznamov t.j. 370 x 12 000 = 4 440 000 B t.j. cca. 4,23 MB

AuthorComment (Id[int], userId[Smallint], authorId[Smallint], userComment[Varchar(510)], ranking[Tinyint], commentDate[Date])
Sum = 4 + 2 + 2 + 510 + 1 + 8 = 527 B
Predpokladá sa 600 užívateľov 10 000 autorov, najhorší možný prípad, keď bude každý autor komentovaný každým užívateľom : 600x10 000x527 = 3 162 000 000 B t.j. cca. 3 GB
To je príliš veľa z toho dôvodu zavádzam obmedzenie na 50 komentárov a staré komentáre sa budú mazať nie však hodnotenia teda záznami.
Sum = 4 + 2 + 2 + 0 + 1 + 8 = 17 B
Najhorší prípad:
50 x 10 000 x 527 + 550 x 10 000 x 17 = 357 000 000 B t.j. cca. 340,46 MB

BookComment (Id[int], userId[Smallint], bookId[Smallint], userComment[Varchar(510)], ranking[Tinyint], commentDate[Date])
Sum = 4 + 2 + 2 + 510 + 1 + 8 = 527 B
Predpokladá sa 600 užívateľov 12 000 kníh, najhorší možný prípad, keď bude každá kniha komentovaný každým užívateľom : 600x12 000x527 = 3 794 400 000 B t.j. cca. 3,53 GB
To je príliš veľa z toho dôvodu zavádzam obmedzenie na 50 komentárov a staré komentáre sa budú mazať nie však hodnotenia teda záznami.
Sum = 4 + 2 + 2 + 0 + 1 + 8 = 17 B
Najhorší prípad:
50 x 12 000 x 527 + 550 x 12 000 x 17 = 428 400 000 B t.j. cca. 408,55 MB

Loan(Id[Int],Book_Id[Int],User_Id[Smallint],DateOfLoan[Date],DateOfRetunr[Date])
Sum = 4 + 4 + 2 + 4 + 4 = 18 B
Hrubý odhad užívatelia x počet kníh t.j. 600 x 12 000 = 6 000 000 záznamov
t.j. celkovo 7 200 000 x 18 = 129 600 000 B t.j. cca. 129,60 MB

AuthorLadder (Id[Int], year[Varchar(4)], authorId[Smallint], evaluationRank[Smallint], evaluationRanking[Tinyint], popularRanking[Int], popularRank[Tinyint]))
Sum = 4 + 4 + 2 + 2 + 1 + 4 + 1 = 18 B
Hrubý odhad : počet autorov 10 000 x počet rokov, pre ktoré chceme rebríčky uchovávať napr. 15 je 10 000 x 15 = 150 000 záznamov
t.j. celkovo 150 000 x 18 = 2 700 000 B t.j. cca. 2,58 MB

BookLadder (Id[Int], year[Varchar(4)], bookId[Smallint], evaluationRank[Smallint], evaluationRanking[Tinyint], popularRanking[Int], popularRank[Tinyint]))
Sum = 4 + 4 + 2 + 2 + 1 + 4 + 1 = 18 B
Hrubý odhad : počet kníh 12 000 x počet rokov, pre ktoré chceme rebríčky uchovávať napr. 15 je 12 000 x 15 = 180 000 záznamov
t.j. celkovo 180 000 x 18 = 3 240 000 B t.j. cca. 3,09 MB
Celkový odhadovaný objem databáze:
3,09 + 2,58 + 129,60 + 408,55 + 340,46 + 4,23 + 0,01 + 0,02 + 3,00 + 0,13 = 891,70 MB
t.j. cca. 1 GB.

4.7.1. Predpokladané platformi a použité technológie:

- Databáza(Oracle 11g)
- Gui(Adobe Flash webová aplikácia s užívateľský prívetivým dynamickým rozhraním)
- Zvolený programovací jazyk pre implementáciu Action Script v 3.0
- Podporné jazyky využitév projekte: PhP, JavaScript, AJAX, XHTML/CSS, C#.
- Nástroje pre implementáciu: MS Visual Studio 2010 [12], Adobe Flash Builder v 4.6 [13], Oracle SQL Developer v3.0.04, Aptana Studio 2.0, Macromedia DreamWeaver 8.0

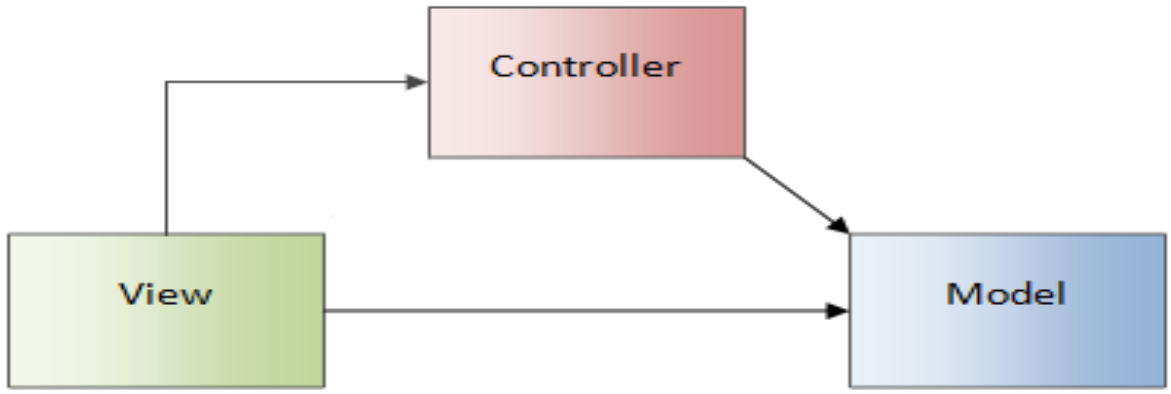
[12] Kniha, v ktorej je takmer všetko o Visual Studiu a jazyku C#, z ktorej som čerpal: **[18]**

4.7.2. Použité návrhové a implementačné vzory:

- **Identity field :** vzor uchovávajúci databázové id v atribúte in-memory objektu a tak zaisťuje správne prepojenie objektu a odpovedajúceho záznamu v databáze.
- **Domain model :** vzor umožňujúci zachovanie dátovej štruktúry a vzťahov medzi objektmi zapúzdrujúci správanie objektu aj dáta do komplexných modelov.
- **Data mapper :** návrhový vzor vytvárajúci vrstvu mapperov, ktoré obsahujú dátovú logiku a umožňujú transformáciu in-memry objektov do databázových záznamov. Všetka logika nutná k načítavaniu, editovaniu, mazaniu a vkladaniu dát do databáze sa nachádza na jednom mieste mimo kódu in-memry objektu, to umožňuje lepšiu štruktúrovateľnosť a prehľadnosť kódu.
- **Transaction script :** vzor zabezpečujúci správanie, kde každá užívateľská akcia je zabezpečená samostatnou procedúrou(využité v controlery).
- **Lazy load:** vzor umožňujúci vytváranie nie úplných objektov a tým znižujúci hardwarové nároky. Objekty nebudú obsahovať všetky dáta v členských atribútoch ale pre niektoré dáta budú existovať metódy, ktoré ich budú v prípade potreby získavať z databáze a ďalej spracovávať.

Prototyp architektúry systému

Zvolený architektonický vzor :
Model-view-controller (MVC [tiež nazývaný Model-2]**)** :



4.8 Užívateľský prístup a optimalizácia:

Výsledný IS bude umožňovať viacnásobný užívateľský prístup, odhadujú sa traja až piati aktívny užívatelia v systéme, v špičke sa predpokladá až dvadsať aktívnych užívateľov. Výsledná aplikácia musí byť rýchla a užívateľsky pohodlná, tomu budú odpovedať aj zvolené hardwarové nároky serveru.
Predpokladá sa skúšobná doba na určenie optimálnych vlastností a nastavení HW&SW.

Prepojenie ASP.Net a db. Oracle je inšpirované **[20].**

---

[13] Kniha v ktorej je takmer všetko o AS 3.0 , z ktorej som čerpal **[19]**

5.0. Záver :

Jazyk ActionScript 3.0 je silne komplexný, adaptabilný, rýchlo sa rozvíjajúci a jednoduchý objektovo orientovaný programovací jazyk, ktorý svojou funkcionalitou a jednoduchosťou môže konkurovať vyspelým objektovo orientovaným jazykom ako sú C# a Java.
Všetky objektovo orientované programovacie princípi sú v tomto jazyku dodržané ba čo viac ešte rozvinuté do vyššej miery ako je tomu u jazykov ako sú Java a C#, viď napr. dva spôsoby dedičnosti v časti „ActionScript 3.0 a dedičnosť“.
Keďže jazyk ActionScript 3.0 je jazykom bežiacim v klientskej časti prehliadača, nezabezpečuje serverovú funkcionalitu a preto ho nie je možno priamo pripojiť k databáze. Tento problém sa dá riešiť prostredníctvom využitia prostredníka, t.j. serverovej technológie, ktorá je tohto schopná.
Jazyk AS 3.0 ponúka sériu prostriedkov, ktoré umožňujú zabezpečenie takej miery integrácie s rozličnými OOP jazykmi, že prepojenie AS a báze dát nie je naďalej problémom. Principiálne aj funkčne je takáto komunikácia zo strany AS 3.0 vždy rovnaká, bez závislosti na zvolenom cieľovom jazyku.
Z toho dôvodu sa problém porovnania a určenia najefektívnejšej metódy presúva na samotné porovnanie jazykov PhP, Java, C# a Visual Basic. Takéto porovnanie je vždy neobjektívne, pretože sa vždy zakladá na určitých subjektívnych kritériách.
Kedže je komunikácia zo strany ActionScriptu vždy rovnaká, je na vývojárovi aby si určil, ktorá z metód uvedených v prechádzajúcich kapitolách je mu najvyhovujúcejšia.

Svoje tvrdenia som si v praxi overil pri tvorbe informačného systému knižnica (viď kapitola 4.0) a za pomoci mnou vytvorených rôznych testovacích aplikácií, ktorých časti ste mohli zhliadnuť v predchádzajúcich kapitolách.

6.0. Použité zdroje a Literatúra:

1. ActionScript. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/ActionScript#ActionScript_3.0_komplexn.C3.AD_datov.C3.A9_typy
2. Objektově orientované programování v jazyce ActionScript. In: [online]. [cit. 2012-04-26]. Dostupné z: http://help.adobe.com/cs_CZ/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7ec0.html
3. Objektově orientované programování v jazyce ActionScript: Pokročilá témata. In: *Http://help.adobe.com* [online]. [cit. 2012-04-19]. Dostupné z: http://help.adobe.com/cs_CZ/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7f3f.html#WS5b3ccc516d4fbf351e63e3d118a9b90204-7f1b
4. Networking and communication: Working with external data. In: *Http://help.adobe.com* [online]. [cit. 2012-04-19]. Dostupné z: http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7cfd.html
5. AS3 LoadVars. In: *Http://www.flashperfection.com* [online]. [cit. 2012-04-19]. Dostupné z: http://www.flashperfection.com/tutorials/Flash-AS3-loadVars-68392.html
6. PASSING VARIABLES FROM HTML TO FLASH VIA FLASH VARS. In: *Http://www.permadi.com* [online]. [cit. 2012-04-19]. Dostupné z: http://www.permadi.com/tutorial/flashVars/indexAs3.html
7. Simple Java Servlet: Simple Servlet Skeleton. [online]. [cit. 2012-04-19]. Dostupné z: http://snippets.dzone.com/posts/show/3489
8. NICKSON, Bobby. AS3 Calling JAVA Servlet. In: [online]. 04.05.2010 [cit. 2012-04-19]. Dostupné z: http://bnickson.blogspot.com/2010/05/as3-calling-java-servlet.html
9. Send / Pass values from Flash as3 to PHP without opening new window. In: *Http://www.designscripting.com* [online]. [cit. 2012-04-19]. Dostupné z: http://www.designscripting.com/2011/09/send-pass-values-from-flash-as3-to-php-without-opening-new-window/
10. Query string. In: *Http://en.wikipedia.org/wiki* [online]. [cit. 2012-04-19]. Dostupné z: http://en.wikipedia.org/wiki/Query_string
11. SADEK, Abdalla. Integrating ASP.NET and ActionScript 3.0 through XML. In: [online]. 21.06.2008 [cit. 2012-04-19]. Dostupné z: http://www.codeproject.com/Articles/27982/Integrating-ASP-NET-and-ActionScript-3-0-through-X
12. SANSANWAL. Post XML Data to an ASP.NET Page using C#. In: *Http://www.codeproject.com* [online]. [cit. 2012-04-19]. Dostupné z: http://www.codeproject.com/Articles/10430/Post-XML-Data-to-an-ASP-NET-Page-using-C
13. Working with XML: XML objects. In: *Http://livedocs.adobe.com/* [online]. [cit. 2012-04-19]. Dostupné z: http://livedocs.adobe.com/flex/3/html/help.html?content=13_Working_with_XML_04.html
14. Working with XML: The E4X approach to XML processing. In: *Http://help.adobe.com* [online]. [cit. 2012-04-19]. Dostupné z: http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7e72.html
15. Sending XML data from AS3 to PHP. In: *Http://www.actionscript.org* [online]. [cit. 2012-04-19]. Dostupné z: http://www.actionscript.org/forums/showthread.php3?t=169554
16. Tutorial: Building a Flash socket server with Java in five minutes. In: [online]. [cit. 2012-04-19]. Dostupné z: http://www.giantflyingsaucer.com/blog/?p=205

17. Tutorial - Connection-Oriented Network Applications: Core functionality. In: [online]. [cit. 2012-04-19]. Dostupné z: http://xsocket.sourceforge.net/core/tutorial/V2/TutorialCore.htm

18. SHARP, John. *Microsoft Visual C# 2010 step by step*. Birmingham: Computer Bookshops [distributor], c2010, 748 s. ISBN 07-356-2670-7.

19. MOOCK, Colin. *Essential ActionScript 3.0*. Sebastopol: O´Reilly, 2007, 911 s. ISBN 978-0-596-52694-8.

20. DOC.ING.KRÁTKY, Michal,Ph.D. a Ing.Radim BAČA,PH.D. VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA. *Výukové materiály predmetu Databázové a informační systémy*. Dostupné z: www.dbedu.cs.vsb

7.0. Prílohy:

DVD 1	Obsah adresára
IsLibraryClient.rar	Zabalená skompilovaná solution klientskej aplikácia pre Adobe Flash Builder v 4.6
IsLibraryServer.rar	Zabalená skompilovaná solution serverovej aplikácie pre MS Visual Studio
Database.rar	Zabalená zložka obsahujúca sql scripty na vytvorenie a naplnenie db Oracle
xSocketAS3.rar	Zabalená testovacia aplikácia socketovej komunikácie pre Adobe Flash Builder v 4.6
xSocketJavaServer.rar	Zabalený Java server pre testovaciu aplikáciu xSocketAS3